(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
23 August 2001 (23.08.2001)

PCT

(10) International Publication Number
WO 01/61438 A2

(51) International Patent Classification⁷: G06F 1/00

(21) International Application Number: PCT/US01/05355

(22) International Filing Date: 20 February 2001 (20.02.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/183,466 18 February 2000 (18.02.2000) US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 60/183,466 (CIP)
Filed on 18 February 2000 (18.02.2000)

(71) Applicant *(for all designated States except US)*: **PERMABIT, INC.** [US/US]; 14 Portland Street, Cambridge, MA 02139 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **MARGOLUS, Norman, H.** [CA/US]; 4 Aldersey Street, #24, Somerville, MA 02143 (US). **KNIGHT, Thomas, F., Jr.** [US/US]; 58 Douglas Road, Belmont, MA 02178 (US). **BOGHOSIAN, Bruce, M.** [US/US]; 6134 Lexington Ridge Road, Lexington, MA 02421 (US).

(74) Agent: **LEE, G., Roger**; Fish and Richardson P.C., 225 Franklin Street, Boston, MA 02110-2804 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: A DATA REPOSITORY AND METHOD FOR PROMOTING NETWORK STORAGE OF DATA

(57) Abstract: In general, the invention features methods by which more than one client program connected to a network stores the same data item on a storage device of a data repository connected to the network. In one aspect, the method comprises encrypting the data item using a key derived from the content of the data item, determining a digital fingerprint of the data item, and storing the data item on the storage device at a location or locations associated with the digital fingerprint. In a second aspect, the method comprises determining a digital fingerprint of the data item, testing for whether the data item is already stored in the repository by comparing the digital fingerprint of the data item to the digital fingerprints of data items already in storage in the repository, and challenging a client that is attempting to deposit a data item already stored in the repository, to ascertain that the client has the full data item.

WO 01/61438 A2

A DATA REPOSITORY AND METHOD FOR PROMOTING NETWORK STORAGE OF DATA

Cross-Reference to Related Applications

This application claims priority from U.S. Provisional Application Serial No. 60/183,466, filed February 18, 2000.

Background of the Invention

For almost as long as there have been computer networks, there have been schemes which allow computers to access each other's file systems over the network in much the same manner as they access their own local file system. The first widely used remote file access protocol was Sun Microsystems' network file system (NFS), which became very popular with the rise of Unix in the mid 1980's (see B. Nowicki, "NFS: Network File System Protocol Specification," Network Working Group RFC1094, March 1989). At about the same time, the SMB network file sharing protocol was developed by IBM for use with their PC's. Subsequent versions of SMB have become widely used on networked PC's running Microsoft Windows, and on their fileservers.

Keeping data in networked file systems allows users to access the same data environment from different workstations on the network, and greatly simplifies system administration and the sharing of public data. For these and other reasons, it is expected that network data repositories will become widely popular among PC users as soon as typical PC network connections become fast enough to make substantial remote storage of data practical. Indeed, some Web-based services which make specific types of user data accessible from any Web browser are already popular -- for example, email services and appointment calendars. Servers for individuals' Web pages also follow the network-data model.

Many companies are offering additional Web-based services which store their data remotely, seeking new applications that will become popular. Some of these companies also offer substantial amounts of free network-based file storage. The greatest obstacle to the acceptance of these new network-based services has been slow network connections. Most computer users currently connect to the network through a telephone modem, which provides them with a connection that is about 1000 times slower than the I/O bandwidth to their local hard disk. This makes it relatively

inconvenient to use remote network-based storage for most of the applications that these users now run on their local file system.

Some companies currently sell network-based backup services to PC users. For a fee, these companies provide a combination of PC software and networked storage space that allows users to keep a copy of their most important data remotely. For privacy, the PC software encrypts user data before sending it to be stored, using the user's individual public key. Some of these companies also offer Web-based access to backed-up data. Thus far, these companies have not achieved an appreciable penetration into the PC user market. Slow network connections, the cost and effort involved in obtaining and using such services, and a low perceived benefit attached to maintaining backups of file data, have been major obstacles. For the moment, most of the Gigabytes of programs and data that users accumulate remain exclusively on their local hard disks.

Use of network storage is also encouraged by techniques which speed up network file transfers. One such technique involves the concept of a "digital fingerprint" of a file, also called a "hash function", a "content signature" or a "message digest" (see R.L. Rivest, "MD4 Message Digest Algorithm," Network Working Group RFC1186, October 1990). A fingerprint is a fixed-length value obtained by mixing all of the bits of the file together in some prescribed deterministic manner -- the same data always produces the same fingerprint. The fingerprint is used as a compact representative of the whole file: if two file fingerprints don't match, then the files are different. For a well designed fingerprint, the chance that any two actual files will ever have the same fingerprint can be made arbitrarily small. Such a fingerprint serves as a unique name for the file data.

Fingerprints have been used for many years to avoid unnecessary file transfers. One application of this sort has been in Bulletin Board Systems (BBSs), which have used fingerprints since the early 1990's to avoid the communication cost of uploading file data that is already present in the BBS, but associated with a different file name. Fingerprints have also been used in BBSs to conserve storage space by not storing duplicate data (for an example of both uses, see Frederick W. Kantor's Content Signature software, FWKCS, which has been in use by bulletin boards such as Channel 1 since at least 1993). These BBSs maintain a table of fingerprints for all files already present. When a new file is uploaded for storage on the BBS, its fingerprint is taken. If

the BBS already contains a file with the same fingerprint (regardless of the file's name) then the duplicate data is not stored. Similarly, a client computer wishing to store data into the BBS can compute the fingerprint of the file that it wishes to send, and send that first. If a file containing this data is already present in the BBS, then the client is informed and need not send anything.

D. A. Farber and R. D. Lachman, in U.S. 5,978,791 (Data processing system using substantially unique identifiers to identify data items, whereby identical data items have the same identifiers, filed October 1997) carry the idea of file fingerprints a step further, using them as the primary identifier for all data-items stored in a file system. In their scheme, not only are fingerprints used to avoid unnecessary transmission and duplicate-storage of file data (as in the BBS scheme mentioned above), but they also use fingerprints directly to gain read access to data. In this scheme, access to "licensed" data is controlled by associating explicit lists of licensees with specific data-items. Such a control mechanism doesn't scale well when applied to intellectual property protection in general. Any data-item added to the system which is copyrighted, for example, would have to have attached to it an explicit list of all users who are legally allowed to read it. Otherwise someone can give out access to the data-item to everyone that uses the file system by anonymously publishing the fingerprint of the data-item. Constructing an explicit legal-access list for each data-item is in general cumbersome, difficult and intrusive.

Furthermore, existing schemes which use fingerprints to identify redundant data and avoid unnecessary transmission and storage depend upon the storage system being able to examine previously stored data. If users independently encrypt their data for privacy, they can't take advantage of each others data to save on transmission or on storage. If data is unencrypted, then the storage system maintainers have complete access to all user data. They may be tempted or coerced into looking at this data, and in some situations may be legally obliged to provide parts of it to third parties.

Summary of the Invention

In general, the invention features a method by which more than one client program connected to a network stores the same data item on a storage device of a data repository connected to the network. The method comprises encrypting the data item using a key derived from the content of the data item, determining a digital fingerprint

of the data item, and storing the data item on the storage device at a location or locations associated with the digital fingerprint.

In preferred implementations, one or more of the following features may be incorporated. The method may further include testing for whether a data item is already stored in the repository by comparing a digital fingerprint of the data item to digital fingerprints of data items already in storage in the repository. The same digital fingerprint may be used for storing the data item on the storage device and for testing whether a data item is already stored in the repository. Encrypting of the data item may be performed by the client prior to transmitting the data item to the storage device. The method may further include encrypting the key and storing the encrypted key on the storage device or on another storage device connected to the network. A client or user specific key may be used to encrypt the key derived from the content of the data item. The key derived from the content of the data item may be the same for all copies of the data item stored in the repository. Users of the method may be grouped into families, and the key derived from the content of the data item may be the same for all copies of the data item stored in the repository by users in the same family, but may be different for users in different families. One or more additional copies or other forms of redundant information about the data items may be stored on the storage device or on other storage devices connected to the network for data integrity, availability, or accessibility purposes and not to provide separate storage of the data item for different client programs. The method may further include associating the data item with each of a plurality of access-authorization credentials, each of which is uniquely associated with a particular user or client program. The method may further include associating the data item with each of a plurality of access-authorization credentials, each of which is uniquely associated with a particular user or client program. Associating of the data item with each of a plurality of access-authorization credentials may include storing a plurality of named objects, each named object comprising information representative of the data item paired with information representative of one of the access-authorization credentials. The information representative of the data item may be a digital fingerprint. The information representative of the access-authorization credential may be a cryptographic hash of all or part of the access-authorization credential. The cryptographic hash may be an access identifier that uniquely identifies the data item for a particular user or client program. The named object may be a data structure created

by the client program. The named object may be a data structure created by a server program acting on behalf of the repository. The method may further include a client replacing an existing version of a data item stored on the storage device with a new version of that data item, by replacing the existing named object with a new named object. The method may further include a client retrieving a data item by accessing a named object using an access-authorization credential to select the named object, and using the contents of the named object to determine the location of the data item on the storage device. The named objects may further include version information associating different data items with different versions of the named object. A backup of data items stored on the storage device may be accomplished by preserving copies of the current versions of named objects in existence at the time of the backup. Data items associated with named objects may not be deleted from the repository, and wherein records are kept of the association between data items and names in order to define named objects, and wherein named objects may be backed up by preserving copies of the named object records in existence at the time of the backup. A backup of data items stored on the storage device may be accomplished by preserving copies of the current versions of named objects in existence at the time of the backup. A plurality of backups may be made at spaced time intervals. The backup may be accomplished by declaring that after a prescribed moment in time a new version of each named object will be created the first time that a new data item is associated with it. The prescribed moment in time is determined separately for each named object. Copies of named objects may be preserved by creating a new version of each named object each time that a new data item is associated with it. Versions of named objects that are deemed unnecessary may be deleted. The determination of which versions of a named object to delete may be based in whole or in part on the times at which the versions were created, and the intervals between these times. The method may further include preparing a digital time stamp of a plurality of named objects to allow a property of these named objects to be proven at a later date. A random or other difficult to guess element may be incorporated into the time stamp hash for each named object, to prevent the property from being proven if this element is deleted. The method may further include determining that a data item stored on the storage device is not referenced by any named object, and reusing the storage space used to store the unreferenced data item. The method may further include altering one or more properties or parameters

associated with an access-authorization credential to change the access rights of a client or user to the data item referenced by that credential. The method may further include a challenge step to ascertain that the client has the full data item. The challenge step may require that the client attempting to store a data item provide correct answers to inquiries as to the content of portions of the data item. The data item content on which the challenge is based may be selected with a degree of randomness. Depositors may use the client to store data items in the repository, and at least some depositors may be required to provide identification upon storing at least some data items. Rules for when a depositor must provide identification may be selected in order to discourage unlawful distribution of access to the data item. There may be a greater degree of user identification or a higher likelihood that user identification will be required when the data item being stored by the depositor has been indicated to be shareable with other users. For a class of data items the items may only be shared if the depositor has provided adequate identification. Identity information about the depositor may be made available to anyone able to access the data item, to discourage unlawful sharing. The identity information may be stored in an encrypted form that the depositor and users subsequently accessing the shared data item can both read. The repository may not have access to the identity information about the depositor. There may be trial users of the repository, and the identity of such trial users may not have not been well verified, but restrictions may be placed on sharing of data items deposited by such trial users. The method may further include limiting access to data items deposited by a poorly verified trial user. Limited access may be provided by limiting the aggregate bandwidth provided for such accesses. Limited access may be provided by limiting the number of simultaneous accesses to the data items. The client may have a directory structure for the data items, the data items may be stored in the repository, and the directory structure may not be evident to the repository maintainers. The client program using the repository may determine which data items to deposit in the repository, and wherein that determination may be based at least in part on the result of a comparison of digital fingerprints establishing that certain data items are not in the repository. Mirroring software may be downloaded to the client using a bootstrap process, wherein a small bootstrap program may be downloaded and executed, and the bootstrap program may manage download and installation of the remainder of the mirroring software. The default for deciding what data items to mirror may be to

mirror all data items. The mirroring may include making a determination of which data items need to be transmitted to the repository, and wherein that determination may be based primarily on a comparison of digital fingerprints for data items at the client and data items in the repository. The access-authorization credential may be determined in part by computing a hash involving elements of the pathname for a file on the client computer. The path name hash may be made unique to a client by introducing a reproducible but randomly chosen element into it. A data item may be represented as a composite of objects, and the component objects may be separately deposited in the repository. Lists of fingerprints for data-items making up a composite data-item may be deposited as an index data item, which can be given an object-name and used for obtaining access to any of the component data-items. A proof-of-deposit may be returned for each component deposit, and the proofs may be presented when the index data item is given an object-name. When transmitting a composite data-item, the client may use fingerprints to avoid retransmitting components following loss of communication. The composite data-item may be encrypted with a key that is only made available to the repository at the moment of access. An email message may be broken up into composite items in such a manner that the individual attachments may be separate component data-items. The physical location at which information about named-objects is stored may be based on access identifiers, to introduce reproducible pseudorandomness into the physical locations of the named-object data. Fingerprints may be determined directly from the data items, and this process produces randomly distributed numbers which can be used to introduce reproducible pseudorandomness into the physical locations of the data items. The repository may give the client a deposit receipt which allows the user to prove that the deposit occurred. An access identifier may be formed to provide proof of ownership of the data item stored in the repository, the access identifier may be formed by producing a one-way hash including identifying information chosen by the client program to identify the data item, and the one-way hash may not be reversed to permit the repository to discover the identity of the client program or user. The identifying information may be associated with the data item on the client. The identifying information may be derived at least in part from the path name of the data item on the client. User-identifying information may be provided to the repository as part of the access-authorization credential. At least some access-authorization credentials may be transferred between users without the use of the

repository. At least one class of users may not be permitted to transfer access using access-authorization credentials.

In a second aspect, the invention features another method by which more than one client program connected to a network stores the same data item on a storage device of a data repository connected to the network. The method comprises determining a digital fingerprint of the data item, testing for whether a data item is already stored in the repository by comparing the digital fingerprint of the data item to the digital fingerprints of data items already in storage in the repository, and challenging a client that is attempting to deposit a data item already stored in the repository, to ascertain that the client has the full data item.

In preferred implementations, one or more of the following features may be incorporated. The challenging may require that the client provide correct answers to inquiries as to the content of portions of the data item. The data item content on which the challenge is based may not easily be predicted by the user or client program. The data item content on which the challenge is based may be determined by the client program without the aid of the repository. Future access to the data item may be provided by creating an access-authorization credential which can be presented at a later time to prove that the challenge has been met for that data item. Each access authorization credential may be uniquely associated with an access owner. Each access authorization credential may include information sufficient to identify the access owner. The access authorization credential may include a fingerprint. The fingerprint may be different from the fingerprint used for testing whether the data item is already stored in the repository. The access authorization credential may be associated with a fingerprint in the repository. The access authorization credential may be associated directly with the data-item or with a record in the repository that is associated with the data-item. The record in the repository with which the access authorization credential is associated may be an access identifier that is associated with the credential by computation of a one way hash function. The access identifier may be stored in the repository and may be compared with a later hash of an access authorization credential to verify access permission to a named object. The access authorization credential may include information sufficient to respond to a challenge. The access authorization credential may include data proof information created during a challenge process that is sufficient to prove to the repository that the challenge was passed. This data proof

information may include the actual challenge response, so that it can be directly verified against the data-item. At least some access-authorization credentials may be transferred between users without the aid of the repository. The usage of some access authorization credential may be restricted for at least one class of access owners. The access authorization credential may only be usable by the access owner. The aggregate bandwidth available to all users of the access authorization credential may be limited. At the time of deposit at least some data items may be associated with a minimum expiration time. At least some data items that expire may be removed and their storage space reused. The repository may keep track of which access owners have deposited a given data item. Upon an access owner informing the repository that a data item is no longer needed, the data item may be deleted or the expiration of the data item may be accelerated. The repository may truncate the list of depositors associated with a data-item, and may never accelerates the expiration of this data item. The method may further include encrypting the data item using a key derived from the content of the data item. Encrypting of the data item may be performed by the client prior to transmitting the data item to the storage device. The method may further include encrypting the key and storing the encrypted key on the storage device or on another storage device connected to the network. A client or user specific key may be used to encrypt the key derived from the content of the data item.

In a third aspect, the invention features a method by which more than one client program connected to a network stores the same data item on a storage device of a data repository connected to the network. The method comprises determining a digital fingerprint of the data item, storing the data item on the storage device at a location or locations associated with the digital fingerprint, associating the data item with each of a plurality of access-authorization credentials, each of which is uniquely associated with an access owner, and preparing a digital time stamp of a plurality of records associating data-items and credentials, to allow a property of these records to be proven at a later date.

In preferred implementations, one or more of the following features may be incorporated. Preparing the digital time stamp may include forming a time stamp hash, and a difficult to guess or random element may be incorporated into the time stamp hash, to prevent the property from being proven if this element is deleted. All data

items in the repository may be time stamped if they remain in the depository for a sufficiently long time period.

In a fourth aspect, the invention features a method for detecting the relative uniqueness of a data item in a repository of data items stored on a storage device at locations associated with their digital fingerprints. The method comprises determining a digital fingerprint of the data item, and determining (or approximating) the number of users with authorization credentials for the data item.

In preferred implementations, one or more of the following features may be incorporated. The data item may be a portion of the body of an e-mail message, and the method may be used to determine the relative uniqueness of the e-mail message in a large population of e-mail messages to determine the likelihood that the e-mail is spam. A decision as to whether a data item is a virus may be made by comparing the relative uniqueness of both the data item and other data items associated with the same application.

In a fifth aspect, the invention features a method for detecting whether a suspect data item is infected with a virus that has a uniform impact on an infected data item. The method comprises determining a digital fingerprint of the suspect data item, comparing the digital fingerprint of the suspect data item to the digital fingerprints of infected data items known to be infected with a virus that consistently affects the data item in the same manner, and basing a decision that the suspect data item contains the virus based on there being a match between the fingerprint of the suspect data item and one or more of the fingerprints of the infected data items.

In preferred implementations, one or more of the following features may be incorporated. The method may further include collecting and providing usage statistics based on number of pointers to a data item in the repository. The usage statistics may be configured to provide marketing penetration information on the data item.

In a sixth aspect, the invention features a method by which more than one client connected to a network stores the same data item on a storage device of a data repository connected to the network. The method comprises determining a digital fingerprint of the data item, testing for whether a data item is already stored in the repository by comparing the digital fingerprint of the data item to the digital fingerprints of data items already in storage in the repository, and associating with a data item an informational tag which may be read by at least some client programs.

In preferred implementations, one or more of the following features may be incorporated. The informational tag may indicate at least one of the following: whether the data item contains spam, whether the data item contains or is a virus, whether the data item is copyrighted, by whom the data item is copyrighted, what royalty payment is due for the copyright. The method may further include the process of collecting royalties or other payments for use of a copyright on a data item based on the indication of whether a data item is copyrighted. The process may enable voluntary payment of such royalties or payments. At least some of the tags may be encrypted using the same key as for each data item, so that users with the data item can read the informational contents of the tag.

In a seventh aspect, the invention features a method by which more than one client connected to a network may store the same data item on a storage device of a data repository connected to the network, and wherein there is a public data repository and a private data repository. The method comprises determining a digital fingerprint of the data item, testing for whether a data item is already stored in the public repository by comparing the digital fingerprint of the data item to the digital fingerprints of data items already in storage in the public repository, and if the data item is present in the public repository, storing a named object in the public repository associating the client with the data item and relying on storage of the data item in the public repository; and if the data item is not present in the public repository, storing a named object in the private repository and relying on storage of the data item in the private repository.

In preferred implementations, one or more of the following features may be incorporated. The client may store a named object for the data item exclusively either in the public or the private repository. The data items may be widely circulated non-electronic media such as books or music, and the method may further include converting the widely circulated non-electronic media to a standardized electronic version, storing the standardized electronic version as a data item in the repository, promoting the availability of the standardized electronic version to users with the right to have access, whereby the likelihood of the data repository storing multiple, slightly-different electronic versions of the non-electronic media is reduced.

In an eighth aspect, the invention features a method by which a client connected to a network over a lower speed connection may provide higher speed access to a data item for application processing than is possible over the relatively low speed connection to the network. The method comprises determining a digital fingerprint of the data item, testing for whether the data item is already stored in a repository by comparing the digital fingerprint of the data item to digital fingerprints of data items already in the repository, only if the data item is not already in the repository, transferring the data item over the lower speed connection from the client to the repository, the repository being connected to the network over a higher speed connection than the client, making a higher speed connection between an application server and the data repository, executing an application on the application server to process the data item stored on the data repository, and returning at least some of the processed data to the client across the lower speed connection.

In preferred implementations, one or both of the data transfers to and from the client may be conducted in the background while other applications are running on the client.

In a ninth aspect, the invention features a method by which multiple clients browse content on a network such as the Internet. The method comprises each of the multiple clients accessing content on the network via one or more proxy servers, determining the digital fingerprint of an item of content passing through the proxy server, storing the item of content in a content repository connected to the proxy server at a location associated with the digital fingerprint, testing for whether a content data item is already stored in the repository by comparing the digital fingerprint of the content data item to the digital fingerprints of content data items already in storage in the repository, associating a content data item already stored in the repository with an access authorization credential uniquely associated with an access owner.

In preferred implementations, one or more of the following features may be incorporated. The data repository may save substantially all content browsed by the clients, thereby preserving the content after it has been altered or removed from the network. The method may further include granting search engines access to the stored content data items or to information about the number of times that data items have been accessed or how recently the data items have been accessed.

In a tenth aspect, the invention features a method by which a plurality of clients connected to a network store the same broadcast data on a storage device of a data repository connected to the network, wherein the broadcast data comprises a sequence of frames or other fragments. The method comprises determining a digital fingerprint of each fragment, testing for whether the fragment is already stored in the repository by comparing a digital fingerprint of the fragment to digital fingerprints of fragments and other data items already in storage in the repository, having only the client or clients that determine that a fragment is not stored in the repository transmit the fragment to the repository, whereby because all but one or a small number of clients will not have to transmit the fragment to effect storage of the fragment to effect storage of the fragment n the repository, most of the clients are able to store the broadcast data in the repository without actually transmitting a significant fraction of the data to the repository.

In preferred implementations, the broadcast data may be video and the fragments may be frames of video. The encrypting may be performed by cellular automata, and may include dividing a data-item into segments in which at least some bits in each segment are considered to be homologous, transforming disjoint groups of homologous bits by applying a state-permutation operation separately to each group, and changing which bits are considered to be homologous and repeating the process. The arrangement of bits into segments can be expressed as having a spatial interpretation, and the spatial origin of each segment may be shifted in a manner determined by an encryption key, with bits in different segments that have the same spatial coordinates considered to be homologous. An encryption key may be used to determine what state-permutation operation is applied to each group of homologous bits in each step. Coalescence may be used for backup/mirroring in which substantially all of a personal computer's data is backed up in this fashion. The method may provide a mirroring capability for a personal computer, and mirroring software with instructions for carrying out the aforesaid steps may be preconfigured on the personal computer upon purchase. The method may provide a mirroring capability for a personal computer, and mirroring software for carrying out the method may be initially configured to mirror essentially all data on the user's computer. The method may provide a mirroring capability for a wireless network device.

In an eleventh aspect, the invention features a method for selling a backup service for backing up or mirroring data on a client computer. The method comprises accepting an unlimited amount of backup or mirroring data from a plurality of client computers, and storing the data in one or more repositories to which the client computers are connected via a network, for free or at a charge substantially less than sufficient to cover the cost of operating the backup service, charging a substantial fee, greater than the fee charged for accepting the data, for recovery of the data from the repositories.

In preferred implementations, one or more of the following features may be incorporated. The fee charged for recovery may be greater when the recovered data is provided quickly, either by express delivery of media containing the data or by delivery over a high-speed data connection. The recovery of data over a slow-speed data connection may be provided at no fee or at a charge substantially less than sufficient to cover the cost of operating the backup service. Data coalescence using digital fingerprints may be used to reduce the amount of data transmitted and stored during backup or mirroring. A charge may be made to third parties for high-speed network access to the client data resident on the repositories.

Other features and advantages of the various aspects of the invention will be apparent from the following detailed description and from the drawings.

Description of the Drawings

FIGURE 1 is a block diagram depicting a user's query to the repository to determine if data is present, and transmit it if necessary.

FIGURE 2 is a block diagram depicting the creation of a named object to secure future read access to a data-item.

FIGURE 3 is a block diagram depicting a read operation using a named object.

FIGURE 4 depicts how a mirroring client can be downloaded and run on a user's computer with very little effort, time or user supervision.

FIGURE 5 depicts the data-item encryption process, which produces an encrypted data-item that is user-independent.

FIGURE 6 depicts a way to allow a user to prove ownership of a named-object, without requiring the repository to hold information from which it can identify the user.

FIGURE 7 illustrates the steps involved in depositing a composite-item and associating it with a named-object.

FIGURE 8 illustrates the steps involved in reading a portion of a composite-item.

FIGURE 9 is a block diagram depicting a user's request that the repository modify a named object to point to new data in the storage.

FIGURE 10 is a block diagram depicting an embodiment of the repository's timestamping service.

FIGURE 11 is a block diagram depicting an encryption scheme based on a reversible cellular automaton.

Detailed Description

This invention deals with the organization and operation of a network-based data repository and an associated data services business. This organization and method of operation are designed to make it both feasible and attractive for computer users with slow network connections to store a copy of their local file system data in remote network-connected storage. The same repository organization is also designed to provide efficient storage and data transmission for users with high-bandwidth network connections. This organization addresses feasibility and attractiveness not only in technical matters, but also in societal and legal matters, such as privacy and copyright.

The envisioned data repository consists of a set of data storage devices connected to the Internet, along with the hardware and software that link them together. These storage devices are arranged in groups at widely separated geographical locations, in order to minimize the impact of localized disasters, and to also minimize network congestion. Erasure-resilient coding techniques operating over the network are used to ensure that data is never lost (see the April 1989 paper by Michael O. Rabin, "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance" in the Journal of the ACM, Volume 36 number 2, pages 335--348).

This repository is unusual in that, like the BBS systems cited above, from a logical standpoint it contains only a single copy of each data-item stored in it no matter how many repository clients (i.e., computers running software acting on behalf of human users) store files into it containing the same data-item. Any replication of data is done purely to assure data integrity (i.e., to make sure data is correct) and to improve

data availability (i.e., to make sure a copy of the data is available) and accessibility (i.e., to make sure data can be accessed reasonably quickly). A pointer to a data-item already contained within this repository can be constructed directly from a copy of the same data-item present on a client computer, without the aid of the repository data-servers. Such pointers can be communicated to the repository in place of the actual data-items themselves.

The unusual organization of the repository is a key element in making significant network storage practicable for computers with slow network connections. Advantage is taken of the fact that most of the data on a typical computer duplicates data that is also present on other machines: operating system files, applications, and data files that have been downloaded over the network or copied from removable media. In order to transfer such files to the repository, client software will typically only have to send a pointer, since the repository will already contain a copy of the data, sent earlier by some other client. An important element in the scheme is arranging to share data in this manner without compromising the privacy of user data -- this is accomplished by sharing encrypted data.

This is a key difference from prior art. Previous schemes have used digital fingerprints (hashes) to avoid communicating data already present at the destination. In the present scheme, the data that is communicated is first encrypted. The encryption is performed using a key derived from the data itself, and this key is never seen in an unencrypted form by the repository servers. Since independent client programs encrypt the same data-item in the same manner, fingerprints can be used to avoid duplicate communication. Unique data is automatically encrypted in a unique manner. Data-items with a length comparable to the fingerprint may be encrypted conventionally without much affect on bandwidth usage or storage. This alleviates concerns that short data-items may be decrypted by guessing them.

To further allay privacy concerns, the repository is careful to avoid storing information that is sufficient to identify who has access to a particular data-item. Additional information provided by user access credentials allows a link to be created transiently at the moment of access. This means that common data-items (such as components of popular programs) can't be traced back to their owners using data present in the repository alone. This also avoids some legal issues associated with subpoenable records.

A major concern for a widely used data repository is to avoid becoming entangled in intellectual property disputes. For example, the Farber/Lachman scheme discussed earlier doesn't deal adequately with the issue of copyright. Unless all copyrighted items are individually identified and labeled with all legal accessors, the scheme fails to protect copyright. The fingerprint of an unlabeled data-item can be broadcast anonymously, giving everyone receiving the broadcast read access to the data-item. In this scenario, the repository company would be unable to point to a responsible party other than itself. The present scheme ensures that there is always a responsible party when access is broadcast: it precludes anonymous broadcast of access. For example, assume that a client has a data-item, and wants to secure future access to a copy of this data-item which it determines, using fingerprints, is already present in the repository. That is, the client wishes to deposit the data-item into the repository without retransmitting it. The repository must determine that the depositor has more than just the fingerprint, because that could have been broadcast anonymously. It therefore challenges the depositor, asking for a small amount of information (such as a specified hash) that proves that the depositor has a copy of the full data-item, before giving the depositor access to the repository's copy of the data-item.

The initial applications contemplated for this repository are mainly archival: storing the complete contents of file systems, mirrored and available live on the network, with historical versions of files also available. The longer term applications center on the role of the repository company as a responsible party in a storage transaction marketplace. By implementing protocols that assure data integrity, persistence, privacy, accessibility and access control, and by using a scheme that avoids certain kinds of legal liability and copyright difficulties, the repository company is poised to help enable a storage transaction marketplace.

Initial Applications

In order to attract a significant volume of data from users with slow network connections, it is not only necessary to lower technical barriers, but also necessary to provide significant positive incentives. While these users can deposit much of their data quickly into the repository, they can only retrieve the actual data-items rather slowly -- it isn't practical for them to use the repository in place of their local hard disk.

There are, however, two practical services that can be provided which justify their depositing substantial amounts of data into the repository: file system mirroring and file system backup.

File system mirroring involves maintaining an up-to-date "mirror" copy of a user's file system within the repository. This mirror constitutes a remote network-based backup version of the local file system in a format which allows immediate network-based access to this data. To achieve this, client software is provided that runs on the user's computer and communicates with the repository data-server, automatically sending information to the repository about files that have changed. This program needs little or no configuration, and uses the client computer's processor and network resources only when they are not needed by other programs. It also performs other useful services, such as checking files for viruses. Once a copy of user data has been deposited in the repository, it is guaranteed to be safe from mishap or malicious mischief, and this data is available for use by its owner from anywhere on the network -- available at all times and with high bandwidth. Some of the files mirrored in the repository could be deleted from the local file system, to save space. If a user has several PC's, all of their data that is scattered among their various machines becomes commonly available through the repository. Mirroring can also be applied to many non-PC devices (e.g., wireless personal digital assistants), further helping to consolidate user data. The owner of the mirrored data can also make their data accessible to network based applications and services: for example, portions of it can be served as Web pages, or copied directly to other network file systems. Third-party Application Service Providers (ASPs) can be given access by the users to portions of their data: for example, a system-maintenance ASP could check for software version incompatibilities in a user's data. Specific software ASP's could allow network-based versions of their software to operate on users' text and presentation documents.

Backup is performed on all repository data, including file system mirror data. The repository data server preserves historical copies of all repository data. These copies also reside in the repository but take up little space, since data-items in the repository are never actually replicated -- only the metadata that associates names with data-items is actually copied. As files change, data-items which are no longer associated with any file (or backup copy of a file) may be erased from the repository, and their storage space reclaimed. For low-bandwidth users, there is little reason to

ever remove any of their backup files from active storage in the repository -- this data is always available. Users are able to retrieve past versions of file data. The repository data-server also periodically time-stamps file system "hash" information using digital timestamp techniques (see S. A. Haber and W. S. Stornetta, Jr., US Patent USRE034954, "Method for secure time-stamping of digital documents," May 30, 1995), allowing the repository to provide incontestable legal evidence that a user had a particular file with particular contents in their file system on a given date.

For users with low-bandwidth and intermittent connectivity to the network, the repository business can provide many inducements to convince them to deposit their data in the repository, aiming to retain them as customers when their connectivity improves. In addition to lowering technical barriers, providing useful services, and guaranteeing data privacy, the business can also offer most or all services to these users for free. They are expected to soon turn into higher-bandwidth constant-connection users, who's continued patronage will provide revenue. Revenue can also be derived from ASPs providing data services to these users, particularly if the repository evolves from a data-mirror into a primary data-storage vehicle. An attractive feature of the repository in this context is that it provides safe and secure storage under the control of the end-user (unlike current network based applications such as Web-based email, which lock the user data into the ASPs database). The repository business can also expect to earn revenue from the traffic at the Web portal that users use to control their repository services and to subscribe to new services. Another potential revenue stream for both the business and the users would involve selling application usage information. Users would be paid who are willing to allow the client software to report such information. For example, information about cross-correlations between the presence of different application programs and data files in the same user's file system would be of great interest to software vendors, particularly if tied to a user name.

The Data Repository

The data repository is a distributed aggregate of data storage devices connected to the network, which together maintain a collection of data-items in a single logical address space, indexed by "datanames" (digital fingerprints) generated directly from the data-items themselves. Logically only one copy of each distinct data-item is kept in the repository, which allows for great economy in use of storage space. In practice, some

redundancy is needed in order to assure data integrity, and to increase data availability and accessibility. Economical transmission of data-items which reside within the repository can be accomplished by sending the dataname in place of the data-item. This is illustrated in Figure 1.

For each data-item 3 that a data-client 1 wishes to deposit into the repository, a cryptographic hash function (digital fingerprint) is calculated from the data-item -- this is the repository dataname 3a for that data-item. Ideally, a cryptographic hash function is a fixed random mapping between arbitrarily long input bit-strings and a fixed-length output. With enough bits in the output value, such a hash is probabilistically "guaranteed" to provide a unique dataname for every distinct data-item that will ever be sent to the repository. In this discussion it will be assumed that the repository uses a well studied public-domain hash function called SHA-1, although other choices would do as well (see National Institute of Standards and Technology, NIST FIPS PUB 180-1, "Secure Hash Standard," U.S. Department of Commerce, April 1995.) This function produces a 20-byte value. It is at present computationally infeasible to find two distinct data files that have the same SHA-1 hash value -- this prevents users from intentionally confusing the repository. If it ever becomes necessary to change the hash function used to index new data-items, old datanames can still be used to retrieve old data.

To deposit a data-item 3 into the repository, the dataname 3a is first used to check whether or not the repository already contains a copy of the data-item. The data-client 1 communicates with the repository data-server 2, asking whether a given dataname 3a corresponds to an existing repository data-item. If not, the data-client sends the data 3. The repository data-server 2 independently recomputes the dataname 3a by hashing the data-item received, in order to verify correct transmission, and to avoid any danger of associating the wrong dataname with a given repository data-item. Once a data-item is in the repository, it never needs to be sent again by anyone (unless it has been removed).

Named Objects

Although repository data-items are written directly, in the primary embodiment of this invention they can only be read indirectly, by referring to "named-objects" such as 10 and 12 in Figure 2. This property is not shared by the scheme of Farber and Lachman mentioned in the background section. This restriction is imposed for several

reasons. First of all, this provides a mechanism for associating a fixed name with changing data: reading the same named-object, different data-items are retrieved at different times. Secondly, this level of indirection is used to implement an access control mechanism for shared data: it is useful to control access to a named-object (e.g., file), rather than to a particular string of bits (i.e., data-item). By associating access-control information with named-objects, restrictions can be placed on which users are allowed to read particular named-objects in the repository. Finally, if the repository handles the creation and modification of the named-objects, then it can tell if a particular data-item is currently associated with any named-object: this makes it possible to identify unreferenced data-items and reuse their storage space.

For these reasons, the repository maintains a named-object database. After ensuring that a data-item 3 being transmitted resides in the repository, the client 1 communicates with the data-server 2 in order to associate the data-item 3 with a named-object 3d (Figure 2). It is possible for the data-server 2 to require that the claim submit a "dataproof", i.e., verify that the client actually has a copy of the data-item 3 being transmitted (and not just a dataname provided by some outside agency) before granting repository read access by associating the data-item 3 with the named-object 3d. A read client 5 (Figure 3) associated with client 1 can use the access-authorization credential 3b that was generated in the deposit transaction to subsequently read data-item 3 indirectly by reference to named-object 3d, but no client can directly read data-item 3. All clients which read using named objects (such as 3d and 10) that are associated with the same dataname 3a actually share access to a single repository data-item 3.

If the client 1 (Figure 2) transmits the data-item 3 to the repository using the dataname 3a only, the data server 2 might, for example, randomly select a few data-bytes belonging to the data-item 3, and request that the client 1 send these to it as a dataproof 3c before associating the named-object 3d with the data-item 3, which will allow future read access. Alternatively, the data-server 2 might select a hash function, and ask the client 1 to send it the value of that function applied to the data-item 3 as the dataproof 3c. Such verification could be routinely performed, or might only be used in extraordinary circumstances, such as in connection with proprietary data-items for which the datanames have been unlawfully broadcast.

When verification of ownership is required, this could also be accomplished in an offline fashion -- allowing the individual client to determine what it needs to prove

for each data-item without directly communicating with the repository. With offline dataproofs, the dataproof 3c in Figure 2 could have been precomputed offline long before the "create-access-credential" request is sent -- the client would have the dataproof 3c ready and waiting when it is needed and wouldn't even need to wait for it to be requested.

To prevent access to datanames which have been anonymously broadcast, an offline dataproof should depend on both the client and the data-item. One way to arrange this is to have a different "challenge-randomizer" value associated with each client -- known to both the client and the repository. The challenge for a given data-item 3 could then be derived in a deterministic fashion using the challenge-randomizer and the data-item itself. A simple way to do this would be to hash together the challenge-randomizer and the dataname 3a and use the result as the seed for a random number generator which selects a set of data-item bytes to be returned; or alternatively just compute a hash on the data-item 3 that depends on the challenge-randomizer. The latter approach has the property that the entire data-item 3 is needed to compute the result of the challenge 3c, and so one party being asked to compute a challenge result on behalf of another would have to be given the challenge-randomizer value. Depending on how this value was selected, this might identify the party trying to gain access, or give away some valuable secret of theirs.

Transmitting Read Access

A client desiring access to a particular named-object 3d transmits its request to a client 5 (Figure 3) that already has access, and the latter client passes along the request (along with the requester's access control information) to the repository data-server 2. If the requester is to share an existing named-object 3d (so that if anyone changes which data-item or data-items are associated with it, the requester will see the change) then the requester is given access to the existing named-object 3d. This kind of "access" transaction is used, for example, to share files. If, instead, the requester is only being given access to the data-item 3 currently associated with the named-object 3d (and will not see any future changes in this named-object) then the data-server 2 will make a new named-object 10 for the requester, associated with the same data-item 3. This kind of "copy" transaction is used, for example, to pass data "by value" to a network-based compute server. In either case, the data-item 3 itself is not copied --

only control information associated with the named-object 3d is replicated in order to communicate data access.

Access could alternatively be transmitted in an offline fashion, by transmitting the named-object access-authorization credential 3b that users require to access the data item 3 themselves (perhaps augmented with other authorization credentials). By including a user-identifying token as a necessary part of the access-authorization credential 3b, the unauthorized broadcasting of access to proprietary data can be discouraged. Thus to cause the repository to make a copy of a named object, a client would need to provide its own authorization information for creating a named-object, along with the access-authorization credentials needed for reading the named-object.

Data-items could also be transmitted directly from one repository user to another using the repository as a kind of data-item compression aid. If the data-source wishes to send a data-item 3 which has been deposited in the repository and to which it has read access, then it only needs to send enough information to the data-recipient to allow it to deposit the data-item 3. This consists of just the dataname 3a, along with whatever information 3c is needed to answer the verification challenge that the recipient must meet in order to deposit by dataname. This form of peer-to-peer copying can be discouraged or controlled by making the verification challenge involve the entire data-item (requiring the source to read the entire item before it can transmit access), and by making the information needed to answer the challenge reveal information about the recipient to the source.

Repository users can grant access to their data to whomever they please by giving them appropriate access authorization credentials and decryption keys. Third parties connected to the network can be granted the access needed to act on behalf of repository users, providing useful applications that manipulate repository data, and performing useful data management and data transformation functions. File systems, databases and other persistent object storage systems can be built by third parties, or by users themselves, on top of the repository named-object mechanism. For example, for maximum privacy client software can maintain its own file system directory data for files kept in the repository, using ordinary encrypted data-items to hold the directory information. The repository itself is simply a secure data store, which avoids unnecessary redundancy in the transmission and storage of data, provides access control, and promises to keep verifiable copies of old data and never lose data.

File system Mirroring

The structure of the repository makes it feasible for a computer user with a low-bandwidth connection to the network to maintain a copy of a local file system in remote storage. This copy appears on the network as a "mirror" file system, which reflects the current state of the user's local file system.

The principal benefits of file system mirroring are data security and data accessibility. Once data is deposited in the repository, it is protected from accidental or malicious loss, and past versions of files are made accessible, certified and time-stamped. Moreover, repository file systems can be accessed at high bandwidth, and from anywhere on the network. Mirrored file system data can, for example, be processed by high-performance network based compute-servers, served as Web pages, retrieved through a Web-browser interface, or "mounted" and used as if it were on a local disk.

The benefits of mirroring a local file system provide justification for low-bandwidth users to keep substantial amounts of data in remote storage. The structure of the repository makes this prospect feasible for such users, by avoiding the need to deposit data which is replicated on more than one local file system. If the complete file system is not mirrored, the repository structure also makes it easier to identify which files should be omitted from the mirror: only unique data-items need to be transmitted to the repository, and so only unique data-items need to be considered for omission.

In addition to providing many benefits, file system mirroring also presents a potential threat to privacy. Users may be reluctant to place a copy of their most private files outside of their physical control. Conversely, the repository maintainers may be reluctant to accept the legal liability of having access to valuable secret files, and even to evidence of criminal activity. These kinds of problems are avoided if it is demonstrably impossible for the repository maintainers to understand any of the mirror data that is sent to them. This can be arranged by using encryption techniques, as is discussed in detail in the next section. Since the mirroring client only needs to write data and never needs to read data, as an additional safeguard the mirroring client can be given only the encryption keys needed to write data, but not those needed to read data. This protects users from having everything that was ever on their computer's disk visible to an antagonist who captures their computer. In order for users to be confident

that appropriate encryption is being used and that no private information is being reported, the source code of the mirroring client software can be openly published. Open publication of mirroring clients also makes it easier for third parties to write their own clients which make use of the repository in novel ways.

Considerations related to setting up mirroring are depicted in Figure 4. In addition to dealing with privacy issues through encryption, the mirroring software is also confronted with smaller barriers that might cause users to abandon mirroring, or not try it in the first place. This is important, since the perceived benefits of mirroring may not be enormous for the typical user; after all, most personal computer users don't currently perform any sort of backup on their data. The first barrier to running the mirroring software 13 is downloading it. This process can be made very short: since the client is designed to talk to repository servers (such as 16), only a minimal "bootstrap" program needs to be downloaded and installed initially, probably by clicking once on a Web page 14. This bootstrap program can download the rest of the client software later on.

Complex program configuration would also discourage use. By default, the client software can be configured on installation to simply mirror everything. Once installed, the function of the client program 15 is to run continuously, checking whether files have changed since they were last mirrored, checking if new file data is already present in the repository, depositing data-items as needed, and maintaining repository directory information. By default, this should all be done in an invisible fashion. While the processor is being heavily used for other tasks, this program should stop running. If other programs are using the network, their outgoing data packets should get priority. Running the mirroring client program should not perceptibly slow down the computer's performance on other tasks.

The perceived benefit of running the mirroring client can also be increased if it has system-health-enhancing properties. It can, for example, check for viruses as it examines the local file system. The client's virus information can be kept up-to-date as it communicates with the repository.

Privacy Through Encryption

To avoid the need to transmit and store common data-items multiple times, all data-items are kept in a single shared data-pool in the repository, indexed by

datanames, as discussed above. Without further elaboration, this arrangement has the drawback that sensitive private data is visible to the repository maintainers. To avoid this problem, files are ordinarily transmitted to the repository in encrypted form. For example, all mirrored file data is encrypted, as is indicated in Figure 4. If the encryption was user-dependent, then each user's encrypted version of the same file would be different, and each user would have to transmit their distinct version of each file. In order to have all users with the same file produce the same encrypted data-item, all files are encrypted in a user-independent fashion: the encryption key for each file is derived from the file data alone. This is depicted in Figure 5.

The procedure for file system mirroring is otherwise the same as discussed above. Each file 17 is compressed and encrypted before computing its dataname 19, which is used to determine whether or not the encrypted data-item 22 needs to be sent to the repository. The client software encrypts files using a datakey 18 that is derived by hashing the compressed file data. To maintain privacy, care is taken that the data repository never sees this datakey "in the clear." For compatibility with media such as audio and video data which are often used in a sequential or streaming fashion, both the compression and the encryption can be performed in a fashion which allows the data-item 22, when being read, to begin to be decrypted and decompressed before the entire data-item has been read.

When a client wishes to retrieve and decrypt a repository data-item, the datakey that was used to encrypt it is needed. For this reason, it is natural to include an encrypted copy of the datakey 20 as part of the named-object in the repository that is associated with this data-item. The encrypted datakey 20 belongs with the named-object rather than with the data-item because the encryption of the datakey will not be the same for all users -- the key 21 used for this will vary from user to user. By making sure that a mirroring client doesn't have (or quickly loses) the ability to decrypt datakeys that it writes, write-only mirroring clients are enabled. This can be accomplished, for example, using public/private key pairs, with the mirroring client only holding the public keys.

Groups of users who wish to share a set of named-objects (for example, a file system) will also share an "aggregate-key" that is used to encrypt all the datakeys in that set of objects. Care is taken that the data repository never sees aggregate-keys in the clear. When access is transmitted by copying a named-object (rather than by

sharing it), the transmitting user gives the unencrypted datakey directly to the access recipient.

Every repository client needs to provide an access-authorization credential in order to read a data-item associated with a named-object. This credential includes a repository-name or "handle" which uniquely identifies the named-object for that client. For the mirroring client, this handle can be derived by hashing the file system path-name on the client's local file system. In this case, it is sufficient for the client to remember all pathnames in it's directory tree in order to be able to reproduce the handles of all of its files. Thus, for example, part of the mirroring process might involve writing data-items which are directory listings for each subdirectory that has changed. Privacy is enhanced if handles are difficult to guess: this can be accomplished by having each mirroring client remember its own randomly chosen "name-randomizer" value which it uses as part of the hashing process that derives handles from file system pathnames. The hashing process might be, for example: start with the name-randomizer and the first component of the pathname, and hash these together; take the result of this hash and hash it with the next component of the pathname, and so on. This kind of hierarchical construction has the advantage that given the handle for some directory along with pathnames starting at that directory, all of the handles for that directory can be constructed. This may make it more convenient to transmit handle information from one client program to another.

While user-independent encryption provides privacy for data-items that are used by only one user, any shared data-item has a vulnerability: given access to the unencrypted file data for any client which shares the data-item, it is easy to discover which file contains the unencrypted data-item. The concern here is not that it will become possible to decrypt the data-item; the unencrypted version was assumed to be available. The conflict with privacy is that it becomes possible for the repository maintainers to identify shared programs and data that a user has in their file system. For example, the repository maintainers could compute the dataname of a particular version of the executable of Microsoft Word, and monitor all transactions to construct a list of all users who have deposited a copy of this program.

Virtual Private Storage Systems

In the scheme described thus far, the datakey used to encrypt the data-items is derived identically by all users from the unencrypted data-item alone. An alternative to this is to have an additional piece of information used to determine the data-item encryption key, which might be called a family key. All users with the same family key use the same method to derive the data-item encryption key from the data; users with different family keys use different methods. For example, a user might use the family key to modify the datakey described above before using it to encrypt the data, as in

data-item encryption-key = E(family-key, datakey)

where E is itself an encryption operation. This has the advantage that it makes a family of data-items more private. For example, this would prevent the repository maintainers from monitoring whether users in this family have deposited specific known pieces of data, since without the family key the repository maintainers will be unable to generate the same data-items and datanames to compare against. This has the disadvantage, of course, that instances of data-items which would have been identical are now made different, and hence the storage and transmission of these data-items becomes less efficient.

Privacy Through Anonymity

If family keys are not used, or if family keys become known, it becomes possible for the repository maintainers to identify shared programs and data that a user has in their file system, which conflicts with user privacy.

This conflict can be avoided if all transactions with the repository are anonymous, so that it is impossible to tell who has access to a particular data-item. Of course, for users to be truly anonymous, all data communications would have to be forwarded through a third party "anonymizer" so that identifying information doesn't appear in the network data packets received by the repository. Anonymous transactions that the repository wishes to charge money for can be handled using electronic cash techniques (see D. Chaum, A. Fiat, and M. Naor, "Untraceable Electronic Cash,"

Advances in Cryptology CRYPTO '88, Springer-Verlag, pp. 319-327). Alternatively, funds can simply be transferred between non-anonymous and anonymous repository accounts using blind signatures.

Anonymity can, however, be a liability. This is the case in connection with named-objects that are shared by many users. These objects can be shared either by separately granting access to each sharer, or by a number of users all sharing the same access information (or even the same identity). In either case, the prospect of users using the repository to illegally share proprietary data (music, videos, programs, etc.) causes a potential problem for the repository maintainers. A completely anonymous repository is much more attractive for these kinds of activities than a more conventional data repository. It may be advisable, for this reason, to limit anonymity in some manner.

Limiting Anonymity

One approach is to eliminate anonymity altogether. Users simply trust the repository to not accumulate or reveal information about the non-unique data that they have in their file systems. In this case, the less information the repository accumulates, the less it can be coerced into revealing. If the repository avoids storing enough information to link users and data-items, then users have a kind of effective anonymity. Extra information provided only at the moment of access can allow users and data to be linked. At that moment, ownership data associated with a named-object can be generated using a cryptographic hash function in a manner that prevents ownership from being discovered, but allows ownership to be proven.

This is illustrated in Figure 6, which contains some details omitted from Figure 3. In this example, we're assuming that the access-authorization credential 3b for a named-object includes a user-identifying token called a "namespace-ID" 3e. A namespace is simply a group of related credentials belonging to a single user. The access-authorization credential 3b also includes a repository handle 3f, which is unguessable by construction. Read access to a named-object may be transmitted from one user to another without the intervention of the repository (i.e., in an offline manner) by transmitting the access-authorization credential 3b. Control over who has the authority to create or use credentials for a given namespace can be handled separately, or can be encoded in additional credentials.

Regardless of the precise composition of the access-authorization credential, anonymous ownership data can be generated from it by hashing the namespace-ID 3e and the handle 3f together using a cryptographic hash function 30. The resulting access identifier 3d is used to identify a named object in the named object database 6. We equate this identifier with the named object itself (cf. Figure 3). The existence of a named object in the database 6 corresponding to the access identifier 3d proves ownership: this database entry was generated when the data-item 3 was associated with the named object 3d (Figure 2). Because of the one-way nature of the cryptographic hash, and because the unguessable handles are never stored in the repository, it is impossible to invert the hash 30 and determine the namespace-ID 3e from the repository's stored access identifier 3d. Since the repository uses the access identifier 3d to determine the data-item 3 that is associated with the named-object, the impossibility of inverting the hash also hides the connection between data-item 3 and the access-owners (i.e., the users or client programs which have established access-authorization credentials) who are able to read it.

Partial Anonymity

Another approach is to treat shared named-objects differently than unshared ones. If these two categories can in fact be distinguished, then unshared objects can be made completely anonymous, while shared objects have no anonymity: all transactions involving shared named-objects require user identity verification. This leaves the repository in the same position as more conventional repositories with respect to intellectual property issues associated with shared files, and in a better position with respect to the privacy of unshared files.

This approach assumes that it is possible to distinguish between shared named-objects and unshared ones. This will in fact be possible if the sharing of access-information can be prevented, so that all sharing is done through explicit "share" requests to the data-server. In particular, in this approach we wouldn't provide an offline method of transmitting access-information without sharing a user-identity. Sharing access-information can be discouraged by holding those who share such information responsible for whatever use is subsequently made of the shared named-object. It can also be arranged for the sharing of access-information to reveal the true identity of the access owner to all sharers (but not to the repository). To permit access

sharers to know who the access owner is -- without this information being visible to the repository -- access owners can be compelled to store their certified identity information in the repository in an encrypted form which only they and the sharers can read. They can be required to prove that they've done this using a zero-knowledge protocol (for a discussion of zero-knowledge protocols, see U. Feige, A. Fiat and A. Shamir, "Zero-knowledge proofs of identity," Journal of Cryptography, 1: 66-94, 1988). If user authentication requires knowledge of the key used to encrypt the identity information, then all users sharing access information will have it.

By limiting anonymity in other ways, it may be possible to put the repository in a still better position. For example, those who are sharing a set of named-objects could be given access to information about who last modified each object, with this information kept invisible to the repository. The identifying information provided could, for example, be a repository email address, with associated personal information revealed by the repository only under a court order. This organization would allow users to confront each other privately concerning controversial sharing of data before trying to compel the repository to intervene.

Poorly Verified Users

Finally, it should be noted that it may be desirable to support some users who are effectively anonymous not because the repository forgets information about them, but because the repository cannot confirm their identities. For example, it may be desirable not to require users trying out the mirroring client to provide any sort of verification of their identities. In this case, it would still be necessary to prevent such users from using their unverified repository accounts to broadcast proprietary data. This can be accomplished by not allowing repository-mediated sharing of data-items that come from unverified accounts, and by not allowing offline transmission of read access to data-items in such accounts. The total aggregate bandwidth available using the data-access privileges of such an account could also be limited, so that sharing of access information doesn't enable more than a small number of users to simultaneously read data from this account at a useful rate.

Composite Objects

There are several reasons to construct named-objects which are composed out of more than one data-item. For example, a mirroring client running over a telephone modem may take hours to deposit a single very large file which is not already in the repository. If this file is broken up into many smaller pieces, then if the telephone connection to the local ISP is lost before completion of the full transfer, all of the pieces which were successfully transferred will not need to be transferred again. If the connection is regained and the transfer attempt is repeated, the normal repository query protocol will discover which pieces have already been deposited, and they will not need to be sent again.

Similarly, some structured items can be sent more efficiently if they are broken up appropriately. For example, email messages with multiple attachments can be transmitted (and stored) more efficiently if they are split up into several pieces, with the divisions occurring at appropriate attachment boundaries. In general, files with a limited amount of user-specific information can segregate this user-specific information into designated segments, allowing the file to be broken up in such a manner that most segments are common between multiple users. For example, a user-name could be assigned to a variable at the beginning of a file, and the name would not need to appear explicitly again.

Finally, for general use of the repository as a network-attached file system, the division of files into smaller blocks is useful.

To support composite structure, it would be expensive in terms of resource usage for the repository to associate with each client a separate copy of the structure information for every file deposited. For a long video file, for example, the repository might store hundreds of thousands of individual data-items corresponding to individual frames of the video, with a corresponding list of datanames repeated for each client which deposits this object. For this reason, it is logical for lists of datanames which describe larger objects (with perhaps other information included) to themselves be deposited as data-items in the repository. These index-items can then be shared, just as any other data-items.

The steps involved in depositing a composite object using an index-item are illustrated in Figure 7. First the individual data-items 40 are deposited into the

repository as described earlier, sending data to the repository data-server 47 only when the data-item is not already present. Then the ordered list of corresponding datanames 42 is deposited as a data-item 41, called an index-item. Assuming the data-items 40 are encrypted, a list of unencrypted datakeys 46 (in the same order as the datanames 42) is deposited as a data-item 45, called a key-item. Finally, the dataname 41a of the index-item 41 and the dataname 45a of the key-item 45 are associated with a named object 49 in the repository. This involves sending an access authorization credential 43 and (assuming verification is required) a list of dataproofs 44, one for each of the data-items 40. Alternatively, it may be more efficient for the server 47 to return a token at deposit time confirming each deposit of the data-items 40, and use these tokens for ownership verification instead of the list of dataproofs 44: this reduces the amount of work that the server 47 has to do at the moment when the named-object is created. Both the index-item 41 and the key-item 45 are encrypted in a user-independent manner, just as any other data-items. The datakey for the key-item 45 becomes the datakey for the entire composite data-item, and is encrypted privately before being stored in the repository, as discussed earlier. The repository is given access to the datakey for the index-item 41 only transiently, when it needs to look at the index-item.

The process of reading part of a composite object is illustrated in Figure 8. In addition to the read-access authorization credential 43 for the named-object 49, a block number 50 is also supplied. This indicates which dataname (e.g., 42b) in the index-item 41 should be referenced. The corresponding data-item 40b is returned to the user. Note that this scheme preserves the atomic-nature of named-object writes: the current data-item that a named-object accesses is changed in a single operation.

Historical Versions of Objects

For mirroring of personal computer file systems over low-bandwidth and intermittent network connections, there is little need to ever erase any data-items from the repository. For repository users with faster connections, however, it would be unreasonable to try to keep every version of every file. As an extreme example, if a file is rewritten every time a byte is added, by the time the file reaches a Megabyte a total of about half a Terabyte of data will have been written. Keeping all versions of such a file should be avoided, if possible.

In a repository which keeps historical versions of named objects, a choice must be made of which data to keep. This issue can be addressed by using repository snapshots and named-object reference counting. A "snapshot" of a file system which has been implemented within the repository is a complete "backup" copy of all directory data and file data at a particular moment in time. Snapshots are relatively inexpensive to make, since no data-items are ever duplicated in the repository. To copy a set of named-objects, only pointer and property information actually needs to be copied. By periodically taking "snapshots" of all named-objects in the repository, the ability is preserved to retrieve previous versions of the state of all objects at particular times, but not at all times. Data-items which aren't associated with any named-object are not needed in any of these snapshot versions of the files kept in the repository. This is illustrated in Figure 9. When write client 56 associates a new data-item 62 with named object 58, the reference count of the previous data-item 60 associated with named object 58 may go to zero. This means that data-item 60 is unreferenced, and it may be deleted and its storage reclaimed. If data-item 60 was part of any file system snapshot, its reference count would not have gone to zero, and so it would be preserved. Thus keeping count of all references by named-objects to data-items allows an unreferenced data-item such as 60 to be erased without any danger of losing the ability to retrieve snapshotted earlier versions of all files.

Since data-items which are common to more than one snapshot are only stored once, this backup scheme can be classified as "incremental." Doubling the interval between snapshots only makes it possible to reclaim space associated with files that changed during each of two consecutive original intervals. Beyond some correlation time, it is expected that the set of files that change during each interval will be substantially different for each interval, and so little is saved by further increasing the interval. For this reason, shorter-interval snapshots are kept for a finite period, and longest-interval snapshots forever. When the named-objects associated with a short-interval snapshot are erased, storage space occupied by data-items that become unreferenced can be reclaimed.

File system snapshots can be implemented by declaring a moment of time to be the snapshot, and all writes after that moment don't overwrite previous versions of the same file -- the incremental backup is accumulated incrementally. Each snapshot

declares that all named objects that make up the file system start a new version the next time they are written, and the old version is preserved.

As long as the capacity of storage devices continues to grow exponentially, there is (for most users) little need to ever move any old data out of the repository, onto archival media. For example, if the longest interval snapshots are taken every month, and half of the monthly change in a typical user's unique data is the addition of new files, and their unique-data disk usage grows at the same rate as the hardware capacity of disks, then keeping all monthly snapshots in the repository forever only increases the total disk usage by about a factor of two. If unique user data doesn't grow exponentially, then total disk usage also grows more slowly than hardware capacity, although old data becomes a more significant portion of total usage.

A limiting case of the snapshot method is to set the time interval between snapshots to zero. This means that every time a named object is rewritten, a new version is created. Every version of every object is kept. If this results in too many versions of some named objects, then a decision is made to declare some of these versions as being unnecessary, and to delete them. Rather than simply prune versions as they are written based on a global time threshold (the snapshot method), versions may be pruned based on many criteria. Decisions on which versions to delete might depend on separate policy information associated with each object, the relative time intervals between different versions of the same object, and even on global time thresholds.

The data-pruning mechanisms discussed imply a distinction between short-term memory and long-term memory in the repository. This distinction reflects the fact that objects that have changed recently are the ones most likely to change again. Thus in the short-term, data-items are kept in a form that it is convenient (or at least possible) to erase. In the long-term, it may be inconvenient (or even impossible) to forget any data-items.

Forgetting the Meaning

The repository is designed to be able to remember historical versions of file data forever. This can be accomplished using standard techniques such as redundancy and archival media. Files which have been removed from the current version of a repository file system can be restored by copying them from an earlier version.

Historical versions of files which have changed remain available. Hash information about each file system is digitally timestamped, to allow the repository to provide legal evidence of the existence and contents of files at specific times in the past (see Timestamping discussion below).

The indelible character of the repository means that it may be difficult or impossible to destroy all traces of old data even if someone badly wants to. The general use of encryption makes it possible, however, to render selected old data meaningless. The basic idea is that the most essential encryption keys are never stored in the data repository, and so the user is free to forget these keys, making all associated data unintelligible. If data that is to be retained is copied before "forgetting" the rest in this manner, then information can be selectively erased: only a now-meaningless encrypted copy of the forgotten data remains in the repository.

If keys have been shared (more than one person knows them), then past data can be forgotten in this manner only if everyone who knows these keys cooperates. One can always, however, stop sharing future versions of files by simply copying them to a new client file system and no longer using the old client file system. This is really all that can be accomplished with certainty, since once data has been shared one is never certain that someone hasn't secretly made a copy of the data.

Other Access-Authorization Credentials

An access-authorization credential is a credential that may be presented by a client program to a repository server in order to prove that it has authorization to read a data-item. In the embodiment described above, an example of such a credential has been provided (Figure 6):

access-authorization-credential 3b = (namespace-ID 3e, handle 3f)

where the namespace-ID 3e identifies the access-owner, and the handle 3f identifies a named-object 3d belonging to that namespace. A client program attempting to use this credential 3b must demonstrate that it is one of the authorized users of the namespace-ID 3e. The existence of a named object 3d in the repository corresponding to the credential 3b records the right of an authorized client to access the corresponding data-item 3.

This example illustrates the general character of an access-authorization credential: it constitutes proof that access has been authorized, and it includes information identifying the access credential's owner. The latter property is really only needed in a credential which can be used by third parties -- this property then helps prevent anonymous broadcast of access capability. For credentials usable by third parties, control is maintained over who is permitted to create or use credentials for a given namespace-ID.

There may be advantages in having access-authorization credentials which allow direct access to a data-item, without reference to a named object in the repository. This is particularly appealing in connection with objects which have stopped changing. For such static objects, information about the association of data-items with names can be conveniently stored in ordinary data-items, thus reducing the size of specialized named-object databases. The metadata for these named objects would be managed by clients, and would not be directly visible to the repository.

An example of a direct-access credential might simply be the information needed to create an access-authorization credential for a named-object in the repository. In the above example, this would be (see Figures 2 and 6),

direct-access-credential = (namespace-ID 3e, dataname 3a, dataproof 3c)

To use this direct-access credential, one could simply create a named-object in the repository at the moment when read access is required (including submission of the dataproof, as shown in Figure 2 and earlier discussed), then read using the associated credential, and then delete the repository named-object.

For this mechanism to work, one would need to have a way to ensure that the data-item 3 is not deleted from the repository. In the discussion of historical versions of objects, we assumed that data-items which are not referenced by any repository named-object can be deleted, and their storage space reused. This deletion mechanism can be easily modified to accommodate direct access credentials. For example, when client 1 deposits data-item 3 (Figure 2), it could specify a minimum expiration period. If data-item 3 becomes unreferenced by repository named objects, it would not be deleted from the repository until after the latest expiration date specified in any deposit.

Rather than require the repository to create and delete a temporary named object, one could simply allow a direct-access credential to be used directly for reading a data-item. As part of the data-item deposit process, the repository could supply some authentication code or signature to augment the direct access credential, allowing it to be used without requiring the dataproof to always be checked. Retaining the dataproof as part of the direct access credential makes it possible to verify credentials if repository signing keys have been compromised, canceled or are otherwise unavailable.

It may be desirable to allow the repository to delete a data-item as soon as all access authorization credentials which reference it have been declared deleted. To allow this, one could associate a reference counting scheme with the direct access credential. This could be done, for example, by associating a per-depositor record with each data-item whenever a direct access credential is created. When the credential is later declared deleted, the corresponding per-depositor record would be deleted. Since large reference counts are unlikely to ever go to zero, it may be that once the number of depositor records passes some threshold, the data-item can simply be marked as permanent. This would bound the number of per-depositor records associated with each data-item.

Note that even if the challenge set by the repository server as part of the deposit process is nondeterministic, it can still be the case that a dataproof or other deposit-proof information returned by the server in response to the deposit is perfectly deterministic and suitable for use in a direct-access credential.

Finally, note that the direct access credential could be the primary access authorization credential -- it is not dependent on the existence of a repository based object credential.

Timestamping

Figure 10 illustrates one possible scheme for timestamping repository named-object data. This scheme has the useful feature that all historical data is automatically timestamped: the repository can prove the ownership and contents of any version of a named object that has not been deleted. Users are not required to save any extra information in order to support this service. Short-lived versions of named objects are not timestamped.

Each named object is assumed to exist in multiple historical versions. In this case, the access authorization credential for a named object includes not only the namespace-ID 72i and handle 73i, but also a version number 74i, which we'll assume is chosen randomly. As usual, the hash of the access authorization credential is the access identifier 71i used to index the named object database 75.

In this example scheme, the repository timestamps all named-objects which pass a certain transience threshold, allowing proofs to be constructed for any timestamped object of when the named-object existed, what data-item it was associated with at that time, and who had access to it. This scheme also makes it possible to automatically lose the ability to construct proofs for objects which have been deleted from the named-object database 75.

In this illustrative scheme, we assume that the set of all named objects is divided up among a set of repository servers, each of which has authoritative information about a subset of the named objects (this division can conveniently be based on the access identifier). We will describe the timestamping procedure for a single repository server 70 -- the procedure for multiple servers is simply to timestamp each server separately. When a proof is needed, the server responsible for the required portion of the named-object space is identified, and it's timestamp information is used.

The access identifier 71 indexes the named object version information stored in a named-object database 75, which includes the dataname 76. We select a subset of the server 70's named object database 75 to be timestamped: for example, all versions which were created more than one week earlier, but less than two. This selects a subset which is not so recent that many of the versions will be deleted as being unneeded. If, in this example, we only perform timestamps once per week, then it makes sense to only timestamp one week's worth of versions at a time. By timestamping a selected subset of versions at once, it becomes possible to organize the timestamp information in a convenient form.

The actual timestamp record 78 consists of a list of cryptographic hashes 80, one per version selected for timestamping. Each hash includes an access identifier 71i for a version of an object as well as a dataname 76i associated with the version. This entire list is saved in the repository as a composite data-item 78, to be used in the future in constructing named-object existence proofs. The corresponding dataname 78a is published publicly, or sent to a digital timestamping service.

Assume for simplicity that the timestamp list 80 is sorted by hash value. If a proof of existence is ever required for a particular version of an object which is still in the repository, its timestamp hash can easily be located within the timestamp data-item 78 for the relevant repository server 70. The data-block containing the relevant hash, along with the index-block for the entire data-item 78 and the published dataname 78a for the index block, provide all the information needed to prove the time of the relevant hash. (More levels of hierarchical hashing could be used to reduce the size of an existence proof.) The timestamp hash for the particular version of a named object in turn allows proof of the ownership and dataname of the version. The dataname then allows data contents to be proven.

If a user deletes an object record such as the one indexed by 71i from the repository metadata, the corresponding timestamp hash 80i can no longer be used to prove anything. This is because of the inclusion of the random version number 74i in constructing the hash, assuming that all record of this number is erased along with the object record 71i. This is an important privacy feature, since timestamps could potentially be used by an adversary to prove that a particular user had access to a particular data-item, if the dataname 76i and handle 73i and version number 74i could all be reconstructed.

Note that if a direct access-authorization credential is supported, separate provisions would have to be made to have its hash included in the timestamping process. For the reasons discussed above, it would be important to include an unguessable component in this hash. It would be the client's responsibility to maintain a copy of any direct access credential that it may want to later prove.

Deposit Receipts

Deposit receipts play a similar role to time-stamps. Users can ask for and receive immediate proof that a deposit was successful, and that a certain level of persistence has been guaranteed. The repository will not make this guarantee until it has taken steps to actually safeguard the data. The actual receipt could simply be a digitally signed set of access-authorization credentials.

A Uniqueness Oracle

In addition to avoiding unnecessary data transmission, there are other uses which can be made of the repository's status as an oracle which can determine whether or not a data-item is unique. A prosaic example would be to use the repository as a "spam" filter. If users are encouraged to keep their email messages in the repository, with the header information separate from the body of the message, then the repository allows users to detect whether or not an email message that they receive contains unique data. Users might reject non-unique messages as junk mail.

The repository can give information not only on the absolute uniqueness of a data-item, but also on it's relative uniqueness. This ability is based upon the reference counts that are maintained by the repository in order to allow the reclamation of space occupied by unreferenced data-items. These reference counts allow the construction, for example, of better spam filters which don't reject relatively uncommon messages. They also allow the repository to, for example, help find viruses by detecting unexpected levels of uniqueness. If a virus always affects an application in the same manner, then the resulting data-item can be tagged in the repository as virus-infected, and immediately identified when seen. If, on the other hand, a virus has a variable effect, then each virus-infected executable file will tend to be significantly less common than other files associated with the same application.

The ability of the repository to tag a shared data-item with information also opens up other possibilities. For example, the first depositor of a data-item might be presumed to hold the copyright (until otherwise demonstrated), and could tag the item with information about who to pay if others want to use this item. Software vendors could tag data-items corresponding to old versions of their software with information about newer versions. All sorts of reviews and annotations could be attached to data-items, both encrypted and unencrypted. Such services could also be operated by third-parties using databases indexed by datanames. Annotations could be hidden from the repository by encrypting them using the datakey from the data-item being tagged.

Online-information vendors (software, music, books, etc.) may be interested directly in the reference counts corresponding to their (and competitor's) data. These counts could, for example, be normalized by the reference counts of all versions of a particular operating system in order to give market penetration statistics for a software

application. The time development of the reference counts gives information about rate of sales.

A Layered Business Structure

The repository has a layered structure which lends itself to being implemented as several separate businesses. First there is the physical storage layer, which keeps data in safe and rapidly accessible high-volume storage. Next there is the data-server layer, which manages data-item storage and access using datanames and named-objects, and is responsible for historical versioning and time-stamping. On top of the data-server are built file system and data-services layers, which will in turn have additional application services layers built on top of them. Each of these distinct layers can be implemented as separate businesses, with competition possible at each level.

The primary business that is the subject of this invention is the data-server layer. This business provides an interface which allows clients to share storage efficiently, and to avoid redundancy in data transmission. The data-server business can make use of existing network storage companies for physical storage during its startup phase, and such companies provide extra storage capacity that can be rapidly deployed in case of unanticipated demand. The data-server business could also make use of other companies and entities for physical storage in the long run -- it is an independent business.

Separating the companies that build file systems and advanced data-services from the data-server business has significant advantages. First of all there is a separation of liability issues, since data-services companies may be given unencrypted access to data that they are expected to protect and hold proprietary or confidential. If a data-services company wishes to challenge what is allowed under copyright laws, for example, the data-server business is not responsible for this client's decisions about to whom it gives access to data. Furthermore, separating advanced data-services from the data-server business makes it possible for competing companies to all make use of the same repository. This both lowers the barriers to competition, and makes it more likely that the repository will be associated with successful data-services companies.

The file system mirroring service, which is designed to help promote the data-server business among low-bandwidth users, doesn't require any separate network fileservers: this service can be handled directly as part of the data-server business. The

mirrored file systems can be maintained directly by the mirroring-client software using client-maintained directory structures that are stored in the repository along with the data. This arrangement provides maximum privacy for user data, since if the directory information is encrypted, not even the structure of the file hierarchy is visible to the repository. The data can be accessed over the network as if it were a local file system by using a device driver which communicates directly with the data-server.

In the long-run, a repository data-server business is expected to make money by charging to mediate transactions between data-storers, data-services providers, and (perhaps) data-storage providers. Charges would reflect resource usage. In the near-term, the mirroring client provides valuable services which can be directly charged for. It would also be possible to charge only for very specific value-added services, such as disaster recovery assistance using mirrored data.

Other Features

Some individuals and organizations may be unwilling to let any of their private data be stored outside of their direct control. Such entities can still make use of the repository to maintain a mirror and backup of their public data, while they manage their private data themselves. The determination of which data is private and which public can be made using the repository query mechanism: a data-item which is already present in the repository can be deemed public. Such an entity will never transmit more than the verification challenge for a data-item to the repository. If such an entity runs its own isolated version of the repository data-server to manage its private data, then it obtains the benefits of communication and storage reduction, while retaining the repository's privacy advantages relative to the data-server maintainers.

Since datanames are obtained using a cryptographic hash, they provide a natural source of pseudo-randomness to help divide the data-service work evenly among data-servers. For example, if a local data-server doesn't recognize a dataname, it can use a portion of the dataname to help it decide which other data-servers are responsible for having the definitive answer as to whether the repository holds the corresponding data-item. Similarly, access identifiers are pseudo-random, and this can be used to help split up repository named-object information evenly among data-servers.

A rapidly growing trend today is the use of computers and digital media to replace other kinds of media. For example, at current disk prices, a high-quality digital

scan of a typical book (compressed) takes about $1 worth of disk space. A music CD takes a similar amount of disk space. An interesting business opportunity built on top of the data repository is to perform these media conversions for people, putting the result directly into the repository. Such a service is already provided by Mp3.com for music CD's, using a specialized CD repository. In the case of the envisioned business, when multiple users perform the same conversion, the repeaters are instantly given access to the data-item. This not only greatly speeds up the conversion for them, but it also avoids filling the repository with many slightly different versions of the same information. The major issue that needs to be resolved in this context is how to avoid infringing upon intellectual property rights. It has not yet been decided in court, for example, whether it is enough that the user demonstrate that they possess a copy of the item and represent that they own it, in order to give them access to a copy. It seems likely that it would be sufficient for a user to mail the physical item to the conversion business, which would destroy the original and give them digital access to an electronic version.

Although the file system mirroring discussion only considered copying file system data from a client with a slow connection to the repository, it might be useful to such users to also provide the capability of mirroring in the opposite direction. This would be particularly useful if users with slow connections are also permitted to control the transfer of data between network file systems at high bandwidth, including such services as downloading files, applying compute servers to their network data, and even using an instant media conversion service such as the one outlined above. Results of such operations could be deposited at high-bandwidth in a user's network file system within the repository, which is mirrored within the user's local file system. The downloaded files, computation results, etc., would all eventually appear on the user's local disk automatically, being transferred as a background task by the file system mirroring software. User-initiated background copying of data between local and remote file systems would also be supported.

A coalescing repository such as the one described herein is very well suited to capturing broadcast digital data. For example, if a digital video program (digital cable TV, HDTV, satellite, etc.) is broadcast to a large number of repository users, each user only needs to deposit a small fraction of the data (perhaps just one frame each) in order to transmit the entire program to the repository. For example, if users deposit one

frame at a time, starting at about the same time, and with some randomization in the order in which they deposit frames, then the task of depositing the program is automatically partitioned between the users by the repository's query-before-transmit protocol. By greatly spreading out the time period over which a broadcast object is deposited, the degree of synchronicity needed between depositors in order to share the deposit burden is greatly reduced. (Some randomization in the order that each client chooses to deposit frames may also help divide up this task). Ideally the broadcast coalesces back into a single compound data-object in the repository. Because of single-frame errors this won't actually be the case, but most of the frames will coalesce. This kind of broadcast deposit is particularly attractive in conjunction with disk-based program time-shifting hardware, which records broadcasts for later viewing. If all programs recorded are subsequently deposited in the repository, then they remain accessible even after the copy on the recorder's disk has been erased to make room for new recordings. Essentially all programs ever recorded could remain accessible to the user.

Similarly, the Web can be viewed as a digital broadcast medium. Users could permanently cache all Web pages they have viewed in the repository. This could be done, for example, by configuring the user's Web Browser to request that Web pages pass through a repository proxy server before being passed on to the user. Instead of temporarily caching Web data, as a normal proxy server would, the repository proxy server would deposit a copy of the Web page into the repository. By using a proxy server, rather than having the user deposit the pages directly, we avoid having a new Web page travel both to and from the user. All pages ever viewed would remain available and searchable by the user. This would result in the repository accumulating a copy of all Web pages viewed by its users, which would be useful in constructing Web search engines. Users would have an incentive to use the repository proxy server, since it makes their history permanently available to them. If the repository is arranging for retrieved data to be cached for availability, then having their data in the repository is useful to content providers, since it can save them bandwidth (the repository can use standard techniques to check if it has the latest version of a URL).

A novel way of encrypting a data-item, suitable for use in the repository, is to use an encryption key to control a reversible cellular automata (RCA) dynamics. (For a discussion of RCA models, see N. Margolus, "Crystalline Computation," in the book

Feynman and Computation, edited by A. Hey, Perseus Books 1999, pages 267 - 305). A CA-based scheme has the advantage that it can be run efficiently in software and can easily be accelerated in hardware, since the dynamics is local and uniform (see N. Margolus, "A mechanism for efficient data access and communication in parallel computations on an emulated spatial lattice," USPTO patent application, filed August 12, 1999). This is illustrated in Figure 11. In this example, the bit-string 90 to be encrypted can be taken to be the cell data for an n-dimensional CA space, with a plurality of bits associated with each cell. In the illustration, we divide the bit-string 90 into four pieces (90a, 90b, 90c and 90d) which we will call bit-fields. Each bit-field can be interpreted as an n-dimensional array of bits, with a fixed mapping between position in the bit-string and position in the array. In general, bit-fields will be the same size in corresponding dimensions, and bits from each bit-field constitute a cell (e.g., 91i). Data is moved within an emulated space by independently spatially shifting each bit-field, interpreted as an n-dimensional array. An example of shifting for 1-dimensional bit-fields is shown in 92. In general, this kind of shifting can be performed efficiently for n-dimensional bit-fields using the techniques discussed in the patent application cited above. Bits 93a that shift past the edge 95a of one dimension wrap around to the opposite edge 95b as bits 94a, and similarly with bits 93b, 93c and 93d. The shift amount and/or direction can be different in each of a sequence of RCA steps, with the amounts and directions controlled by portions (99a, 99b, 99c, 99d) of the key 99, interpreted as binary numbers. In between data shifting steps, some or all cells (such as 91i) can be updated individually, with invertibility guaranteed by having the operation performed on each cell be a permutation on the cell's state set. The choice of permutation in each such transformation can be determined by bits of the key (such as 99e). If more bits than are present in the key are desired to control the sequence of shifts and permutations, the key may be transformed in some iterative fashion to produce additional control bits.

Other Embodiments

Although some of this discussion has focused on mirroring of file system data, the methods and protocols described here are of much more general utility. File system mirroring is discussed primarily as an initial application, to help establish the repository. As noted above, the operation of the data-servers and their associated data-

transmission and data-storage protocol constitute a separate business which is compatible with a wide variety of clients, and a wide variety of data-storage entities. This business and protocol will evolve with time.

It is to be understood that while the invention has been described in conjunction with the detailed description thereof, the foregoing description is intended to illustrate and not limit the scope of the invention, which is defined by the scope of the appended claims.

Other embodiments are within the scope of the following claims.

What is claimed is:

1. A method by which more than one client program connected to a network stores the same data item on a storage device of a data repository connected to the network, the method comprising:

encrypting the data item using a key derived from the content of the data item;

determining a digital fingerprint of the data item; and

storing the data item on the storage device at a location or locations associated with the digital fingerprint.

2. The method of claim 1 further comprising testing for whether a data item is already stored in the repository by comparing a digital fingerprint of the data item to digital fingerprints of data items already in storage in the repository.

3. The method of claim 2 wherein the same digital fingerprint is used for storing the data item on the storage device and for testing whether a data item is already stored in the repository.

4. The method of claim 1 wherein the encrypting of the data item is performed by the client prior to transmitting the data item to the storage device.

5. The method of claim 4 further comprising encrypting the key and storing the encrypted key on the storage device or on another storage device connected to the network.

6. The method of claim 5 wherein a client or user specific key is used to encrypt the key derived from the content of the data item.

7. The method of claim 1 wherein the key derived from the content of the data item is the same for all instances of the data item stored in the repository.

8. The method of claim 1 wherein users of the method are grouped into families, and the key derived from the content of the data item is the same for all instances of the data item stored in the repository by users in the same family, but may be different for users in different families.

9. The method of claim 2 wherein one or more additional copies or other forms of redundant information about the data items is stored on the storage device or on other storage devices connected to the network for data integrity, availability, or accessibility purposes and not to provide separate storage of the data item for different client programs.

10. The method of claim 1 further comprising associating the data item with each of a plurality of access-authorization credentials, each of which is uniquely associated with a particular user or client program.

11. The method of claim 2 further comprising associating the data item with each of a plurality of access-authorization credentials, each of which is uniquely associated with a particular user or client program.

12. The method of claim 10 wherein the associating of the data item with each of a plurality of access-authorization credentials comprises storing a plurality of named objects, each named object comprising information representative of the data item paired with information representative of one of the access-authorization credentials.

13. The method of claim 12 wherein the information representative of the data item is a digital fingerprint.

14. The method of claim 12 wherein the information representative of the access-authorization credential is a cryptographic hash of all or part of the access-authorization credential.

15. The method of claim 14 wherein the cryptographic hash is an access identifier that uniquely identifies the data item for a particular user or client program.

16. The method of claim 12 wherein the named object is a data structure created by the client program.

17. The method of claim 12 wherein the named object is a data structure created by a server program acting on behalf of the repository.

18. The method of claim 12 further comprising a client replacing an existing version of a named object with a new version of that named object, by replacing the existing association with a data item stored on the storage device with a new association.

19. The method of claim 12 further comprising a client retrieving a data item by accessing a named object using an access-authorization credential to select the named object, and using the contents of the named object to determine the location of the data item on the storage device.

20. The method of claim 12 wherein the named objects further comprise version information associating different data items with different versions of the named object.

21. The method of claim 20 wherein a backup of data items stored on the storage device is accomplished by preserving copies of the current versions of named objects in existence at the time of the backup.

22. The method of claim 1 wherein records are kept of the association between data items and names in order to define named objects, and wherein data items recorded as being associated with named objects are not deleted from the repository, and wherein named objects are backed up by preserving copies of the named object records in existence at the time of the backup.

23. The method of claim 21 or 22 wherein a plurality of backups are made at spaced time intervals.

24. The method of claim 21 or 22 wherein the backup is accomplished by declaring that after a prescribed moment in time a new version of each named object will be created the first time that a new data item is associated with it.

25. The method of claim 24 wherein the prescribed moment in time is determined separately for each named object.

26. The method of claim 22 wherein named objects are preserved by creating a new version of each named object each time that a new data item is associated with it.

27. The method of claim 26, wherein versions of named objects that are deemed unnecessary are deleted.

28. The method of claim 27, wherein the determination of which versions of a named object to delete is based in whole or in part on the times at which the versions were created, and the intervals between these times.

29. The method of claim 20 further comprising preparing a digital time stamp of a plurality of named objects to allow a property of these named objects to be proven at a later date.

30. The method of claim 29 wherein a random or other difficult to guess element is incorporated into the time stamp hash for each named object, to prevent the property from being proven if this element is deleted.

31. The method of claim 12 further comprising determining that a data item stored on the storage device is not referenced by any named object, and reusing the storage space used to store the unreferenced data item.

32. The method of claim 12 further comprising altering one or more properties or parameters associated with an access-authorization credential to change the access rights of a client or user to the data item referenced by that credential.

33. The method of claim 2 further comprising a challenge step to ascertain that the client has the full data item.

34. The method of claim 33 wherein the challenge step comprises requiring that the client attempting to store a data item provide correct answers to inquiries as to the content of portions of the data item, or inquiries that require knowledge of this content.

35. The method of claim 34 wherein the data item content on which the challenge is based is selected with a degree of randomness.

36. The method of claim 2 wherein depositors use the client to store data items in the repository, and at least some depositors are required to provide identification.

37. The method of claim 36 wherein rules for when a depositor must provide identification are selected in order to discourage unlawful distribution of access to the data item.

38. The method of claim 37 wherein there is a greater degree of user identification or a higher likelihood that user identification will be required when the data item being stored by the depositor has been indicated to be shareable with other users.

39. The method of claim 37 wherein for a class of data items the items may only be shared if the depositor has provided adequate identification.

40. The method of claim 38 or 39 wherein identity information about the depositor is made available to anyone able to access the data item, to discourage unlawful sharing.

41. The method of claim 40 wherein the identity information is stored in an encrypted form that the depositor and users subsequently accessing the shared data item can both read.

42. The method of claim 41 wherein the repository is not able to decrypt the identity information about the depositor.

43. The method of claim 37 wherein the identity of some users has not been well verified, but restrictions are placed on sharing of data items deposited by such poorly verified users.

44. The method of claim 43 further comprising limiting access to data items deposited by a poorly verified user.

45. The method of claim 44 wherein the limited access is provided by limiting the aggregate bandwidth provided for such accesses.

46. The method of claim 44 wherein the limited access is provided by limiting the number of simultaneous accesses to the data items.

47. The method of claim 2 wherein the client has a directory structure for the data items, the data items are stored in the repository, and the directory structure is not evident to the repository maintainers.

48. The method of claim 2 wherein the client program using the repository is a mirroring program which determines which data items to deposit in the repository, and wherein that determination is based at least in part on the result of a comparison of digital fingerprints establishing that certain data items are not in the repository.

49. The method of claim 48 wherein mirroring software is downloaded to the client using a bootstrap process, wherein a small bootstrap program is downloaded and executed, and the bootstrap program manages download and installation of the remainder of the mirroring software.

50. The method of claim 48 wherein the default for deciding what data items to mirror is to mirror all or substantially all data items.

51. The method of claim 48 wherein the mirroring comprises making a determination of which data items need to be transmitted to the repository, and wherein

that determination is based primarily on a comparison of digital fingerprints for data items at the client and data items in the repository.

52. The method of claim 10 wherein the access-authorization credential is determined in part by computing a hash involving elements of the pathname for a file on the client computer.

53. The method of claim 52 wherein the path name hash is made unique to a client by introducing a reproducible but randomly chosen element into it.

54. The method of claim 12 wherein a data item is represented as a composite of data-items, and the component data-items are separately deposited in the repository.

55. The method of claim 54 wherein lists of fingerprints for data-items making up a composite data-item are deposited as an index data item, which can be given an object-name and used for obtaining access to any of the component data-items.

56. The method of claim 55 wherein a proof-of-deposit is returned for each component deposit, and some or all of the proofs are presented when the index data item is given an object-name.

57. The method of claim 56 wherein, when transmitting a composite data-item, the client uses fingerprints to avoid retransmitting components following loss of communication.

58. The method of claim 57 wherein the index data-item is encrypted with a key that is only made available to the repository at the moment of access.

59. The method of claim 55 wherein an email message is broken up into component items in such a manner that the individual attachments are separate component data-items.

60. The method of claim 15 wherein the physical location at which information about named-objects is stored is based on access identifiers, to introduce reproducible pseudorandomness into the physical locations of the named-object data.

61. The method of claim 1 wherein the fingerprints are determined from the data items, and this process produces randomly distributed numbers which can be used to introduce reproducible pseudorandomness into the physical locations of the data items.

62. The method of claim 2 wherein an access identifier is formed to provide proof of ownership of the data item stored in the repository, the access identifier is formed by producing a one-way hash including item-identifying information chosen by the client program to identify the data item, and the one-way hash cannot be reversed to permit the repository to discover the identity of the client program or user.

63. The method of claim 62 wherein the item-identifying information is associated with the data item on the client.

64. The method of claim 63 wherein the item-identifying information is derived at least in part from the path name of the data item on the client.

65. The method of claim 62 wherein user-identifying information is provided to the repository as part of the access-authorization credential.

66. The method of claim 65 wherein at least some access-authorization credentials can be transferred between users without the use of the repository.

67. The method of claim 65 wherein at least one class of users is not permitted to transfer access using access-authorization credentials.

68. A method by which more than one client program connected to a network stores the same data item on a storage device of a data repository connected to the network, the method comprising:

determining a digital fingerprint of the data item;

testing for whether the data item is already stored in the repository by comparing the digital fingerprint of the data item to the digital fingerprints of data items already in storage in the repository; and

challenging a client that is attempting to deposit a data item already stored in the repository, to ascertain that the client has the full data item.

69. The method of claim 68 wherein the repository gives the client a deposit receipt which allows the user to prove that the deposit occurred.

70. The method of claim 68 wherein the challenging comprises requiring that the client provide correct answers to inquiries as to the content of portions of the data item, or inquiries that require knowledge of this content.

71. The method of claim 70 wherein the data item content on which the challenge is based is not easily predicted by the user or client program.

72. The method of claim 70 wherein the data item content on which the challenge is based can be determined by the client program without the aid of the repository.

73. The method of claim 68 wherein future access to the data item deposited is provided by creating an access-authorization credential which can be presented at a later time to prove that the challenge has been met for that data item.

74. The method of claim 73 wherein each access authorization credential is uniquely associated with a access owner.

75. The method of claim 73 wherein each access authorization credential includes information sufficient to identify the access owner.

76. The method of claim 73 wherein the access authorization credential includes a fingerprint.

77. The method of claim 73 wherein the access authorization credential is associated with a fingerprint in the repository.

78. The method of claim 76 or 77 wherein the fingerprint is different from the fingerprint used for testing whether the data item is already stored in the repository.

79. The method of claim 73 wherein the access authorization credential is associated directly with the data-item or with a record in the repository that is associated with the data-item.

80. The method of claim 79 wherein the record in the repository with which the access authorization credential is associated is an access identifier that is associated with the credential by computation of a one way hash function.

81. The method of claim 80 wherein the access identifier is stored in the repository and is compared with a later hash of an access authorization credential to verify access permission to a named object.

82. The method of claim 73 wherein the access authorization credential may include information sufficient to respond to a challenge.

83. The method of claim 73 wherein the access authorization credential includes data proof information created during a challenge process that is sufficient to prove to the repository that the challenge was passed.

84. The method of claim 83 wherein the data proof information comprises the actual challenge response, so that it can be directly verified against the data-item.

85. The method of claim 73 wherein at least some access-authorization credentials can be transferred between users without the aid of the repository.

86. The method of claim 85 wherein the usage of some access authorization credential is restricted for at least one class of access owners.

87. The method of claim 86 wherein the access authorization credential is only usable by the access owner.

88. The method of claim 86 wherein the aggregate bandwidth available to all users of the access authorization credential is limited.

89. The method of claim 68 wherein at the time of deposit at least some data items are associated with a minimum expiration time.

90. The method of claim 89 wherein at least some data items that expire are removed and their storage space reused.

91. The method of claim 90 wherein the repository keeps track of which access owners have deposited a given data item.

92. The method of claim 91 wherein upon an access owner informing the repository that a data item is no longer needed, the data item is deleted or the expiration of the data item is accelerated.

93. The method of claim 92 wherein the repository truncates the list of depositors associated with a data-item, and never accelerates the expiration of this data item.

94. The method of claim 68 further comprising encrypting the data item using a key derived from the content of the data item.

95. The method of claim 94 wherein the encrypting of the data item is performed by the client prior to transmitting the data item to the storage device.

96. The method of claim 94 further comprising encrypting the key and storing the encrypted key on the storage device or on another storage device connected to the network.

97. The method of claim 96 wherein a client or user specific key is used to encrypt the key derived from the content of the data item.

98. A method by which more than one client program connected to a network stores the same data item on a storage device of a data repository connected to the network, the method comprising:

determining a digital fingerprint of the data item;

storing the data item on the storage device at a location or locations associated with the digital fingerprint;

associating the data item with each of a plurality of access-authorization credentials, each of which is uniquely associated with an access owner; and

preparing a digital time stamp of a plurality of records associating data-items and credentials, to allow a property of these records to be proven at a later date.

99. The method of claim 98 wherein preparing the digital time stamp comprises forming a time stamp hash, and wherein a difficult to guess or random element is incorporated into the time stamp hash, to prevent the property from being proven if this element is deleted.

100. The method of claim 98 wherein all data items in the repository are time stamped if they remain in the depository for a sufficiently long time period.

101. A method for quantifying the degree of uniqueness of an indicated data item in a repository of data items stored on a storage device at locations associated with their digital fingerprints, the method comprising:

creating access-authorization credentials which permit users or clients to access data-items that they have deposited; and

determining (or approximating) the number of users with access authorization credentials for the indicated data item.

102. The method of claim 101 wherein the data item is a portion of the body of an e-mail message, and the method is used to determine the relative uniqueness of the portion of the e-mail message in a large population of e-mail messages to determine the likelihood that the e-mail is spam.

103. The method of claim 101 wherein a decision as to whether a data item is a virus is made by comparing the relative uniqueness of both the data item and other data items associated with the same application.

104. The method of claim 101 further comprising collecting and providing usage statistics based on the degree of uniqueness of data items in the repository.

105. The method of claim 104 wherein the usage statistics are configured to provide marketing penetration information on the data item.

106. A method by which more than one client connected to a network stores the same data item on a storage device of a data repository connected to the network, the method comprising:

determining a digital fingerprint of the data item;

testing for whether a data item is already stored in the repository by comparing the digital fingerprint of the data item to the digital fingerprints of data items already in storage in the repository; and

associating with a data item an informational tag which may be read by at least some client programs.

107. The method of claim 106 wherein the informational tag indicates at least one of the following: whether the data item contains spam, whether the data item contains or is a virus, whether the data item is copyrighted, by whom the data item is copyrighted, what royalty payment is due for the copyright.

108. The method of claim 107 further comprising the process of collecting royalties or other payments for use of a copyright on a data item based on the indication of whether a data item is copyrighted.

109. The method of claim 108 wherein the process enables voluntary payment of such royalties or payments.

110. The method of claim 106 further comprising encrypting the data item using a key derived from the content of the data item.

111. The method of claim 110 wherein at least some of the tags are encrypted using the same key as for each data item, so that users with the data item can read the informational contents of the tag.

112. A method by which more than one client connected to a network may store the same data item on a storage device of a data repository connected to the network, and wherein there is a public data repository and a private data repository, the method comprising:

determining a digital fingerprint of the data item;

testing for whether a data item is already stored in the public repository by comparing the digital fingerprint of the data item to the digital fingerprints of data items already in storage in the public repository; and

if the data item is present in the public repository, creating an access authorization credential for the public repository associating the client with the data item and relying on storage of the data item in the public repository; and if the data item is not present in the public repository, creating an access authorization credential for the private repository and relying on storage of the data item in the private repository.

113. The method of claim 112 wherein the client creates an access authorization credential for the data item exclusively either in the public or the private repository.

114. The method of claim 2 wherein the data items are widely circulated non-electronic media such as books or music, and the method further comprises converting the widely circulated non-electronic media to a standardized electronic version;

storing the standardized electronic version as a data item in the repository;

promoting the availability of the standardized electronic version to users with the right to have access, whereby the likelihood of the data repository storing multiple, slightly-different electronic versions of the non-electronic media is reduced.

115. A method by which a client connected to a network over a lower speed connection may provide higher speed access to a data item for application processing than is possible over the relatively low speed connection to the network, the method comprising:

determining a digital fingerprint of the data item;

testing for whether the data item is already stored in a repository by comparing the digital fingerprint of the data item to digital fingerprints of data items already in the repository;

only if the data item is not already in the repository, transferring the data item over the lower speed connection from the client to the repository, the repository being connected to the network over a higher speed connection than the client;

making a higher speed connection between an application server and the data repository;

executing an application on the application server to process the data item stored on the data repository;

returning at least some of the processed data to the client across the lower speed connection.

116. The method of claim 115 wherein one or both of the data transfers to and from the client are conducted in the background while other applications are running on the client.

117. A method by which multiple clients browse content on a network such as the Internet, the method comprising:

each of the multiple clients accessing content on the network via one or more proxy servers;

determining the digital fingerprint of an item of content passing through the proxy server;

storing the item of content in a content repository connected to the proxy server at a location associated with the digital fingerprint;

testing for whether a content data item is already stored in the repository by comparing the digital fingerprint of the content data item to the digital fingerprints of content data items already in storage in the repository;

associating a content data item already stored in the repository with an access authorization credential uniquely associated with an access owner.

118. The method of claim 117 wherein the data repository saves substantially all content browsed by the clients, thereby preserving the content after it has been altered or removed from the network.

119. The method of claim 118 further comprising granting search engines access to the stored content data items or to information about the number of times that data items have been accessed or how recently the data items have been accessed

120. A method by which a plurality of clients connected to a network store the same broadcast data on a storage device of a data repository connected to the network, wherein the broadcast data comprises a sequence of frames or other fragments, the method comprising:

determining a digital fingerprint of each fragment;

testing for whether the fragment is already stored in the repository by comparing a digital fingerprint of the fragment to digital fingerprints of fragments and other data items already in storage in the repository;

having only the client or clients that determine that a fragment is not stored in the repository transmit the fragment to the repository;

whereby because all but one or a small number of clients will not have to transmit the fragment to effect storage of the fragment in the repository, most of the

clients are able to store the broadcast data in the repository without actually transmitting a significant fraction of the data to the repository.

121. The method of claim 120 wherein the broadcast data is video and the fragments are frames of video.

122. A method of encrypting a bit-string using cellular automata, comprising

dividing the bit-string into segments in which at least some bits in each segment are considered to be homologous;

transforming disjoint groups of homologous bits by applying a state-permutation operation separately to each group; and

changing which bits are considered to be homologous and repeating the process.

123. The method of claim 122 wherein the arrangement of bits into segments can be expressed as having a spatial interpretation, and the spatial origin of each segment is shifted in a manner determined by an encryption key, with bits in different segments that have the same spatial coordinates considered to be homologous.

124. The method of claim 123 wherein an encryption key is used to determine what state-permutation operation is applied to each group of homologous bits in each step.

125. The method of claim 48 wherein the aforesaid steps of the method provide a mirroring capability for a personal computer, and mirroring software with instructions for carrying out the aforesaid steps is preconfigured on the personal computer upon purchase.

126. The method of claim 48 wherein the aforesaid steps of the method provide a mirroring capability for a personal computer, and mirroring software for carrying out the method is initially configured to mirror essentially all data on the user's computer.

127. The method of claim 48 wherein the aforesaid steps of the method provide a mirroring capability for a wireless network device.

128. A method for selling a backup service for backing up or mirroring data on a client computer, the method comprising:

accepting an unlimited amount of backup or mirroring data from a plurality of client computers, and storing the data in one or more repositories to which the client computers are connected via a network, for free or at a charge substantially less than sufficient to cover the cost of operating the backup service;

charging a substantial fee, greater than the fee charged for accepting the data, for recovery of the data from the repositories.

129. The method of claim 128 wherein the fee charged for recovery is greater when the recovered data is provided quickly, either by express delivery of media containing the data or by delivery over a high-speed data connection.

130. The method of claim 128 wherein recovery of data over a slow-speed data connection is provided at no fee or at a charge substantially less than sufficient to cover the cost of operating the backup service.

131. The method of claim 128, 129, or 130 wherein data coalescence using digital fingerprints is used to reduce the amount of data transmitted and stored during backup or mirroring.

132. The method of claim 128 wherein a charge is made to third parties for high-speed network access to the client data resident on the repositories.

133. The method of claim 68 wherein records are kept of the association between data items and names in order to define named objects, and wherein data items recorded as being associated with named objects are not deleted from the repository, and wherein named objects are backed up by preserving copies of the named object records in existence at the time of the backup.

134. The method of claim 68 wherein a backup of data items stored on the storage device is accomplished by preserving copies of the current versions of named objects in existence at the time of the backup.

135. The method of claim 133 or 134 wherein a plurality of backups are made at spaced time intervals.

136. The method of claim 133 or 134 wherein the backup is accomplished by declaring that after a prescribed moment in time a new version of each named object will be created the first time that a new data item is associated with it.

137. The method of claim 136 wherein the prescribed moment in time is determined separately for each named object.

138. The method of claim 133 wherein named objects are preserved by creating a new version of each named object each time that a new data item is associated with it.

139. The method of claim 138 wherein versions of named objects that are deemed unnecessary are deleted.

140. The method of claim 139 wherein the determination of which versions of a named object to delete is based in whole or in part on the times at which the versions were created, and the intervals between these times.

141. The method of claim 68 wherein depositors use the client to store data items in the repository, and at least some depositors are required to provide identification.

142. The method of claim 141 wherein rules for when a depositor must provide identification are selected in order to discourage unlawful distribution of access to the data item.

143. The method of claim 142 wherein there is a greater degree of user identification or a higher likelihood that user identification will be required when the data item being stored by the depositor has been indicated to be shareable with other users.

144. The method of claim 142 wherein for a class of data items the items may only be shared if the depositor has provided adequate identification.

145. The method of claim 143 or 144 wherein identity information about the depositor is made available to anyone able to access the data item, to discourage unlawful sharing.

146. The method of claim 145 wherein the identity information is stored in an encrypted form that the depositor and users subsequently accessing the shared data item can both read.

147. The method of claim 146 wherein the repository is not able to decrypt the identity information about the depositor.

148. The method of claim 143 wherein the identity of some users has not been well verified, but restrictions are placed on sharing of data items deposited by such poorly verified users.

149. The method of claim 148 further comprising limiting access to data items deposited by a poorly verified user.

150. The method of claim 149 wherein the limited access is provided by limiting the aggregate bandwidth provided for such accesses.

151. The method of claim 149 wherein the limited access is provided by limiting the number of simultaneous accesses to the data items.

152. The method of claim 73 wherein the access-authorization credential is determined in part by computing a hash involving elements of the pathname for a file on the client computer.

153. The method of claim 152 wherein the path name hash is made unique to a client by introducing a reproducible but randomly chosen element into it.

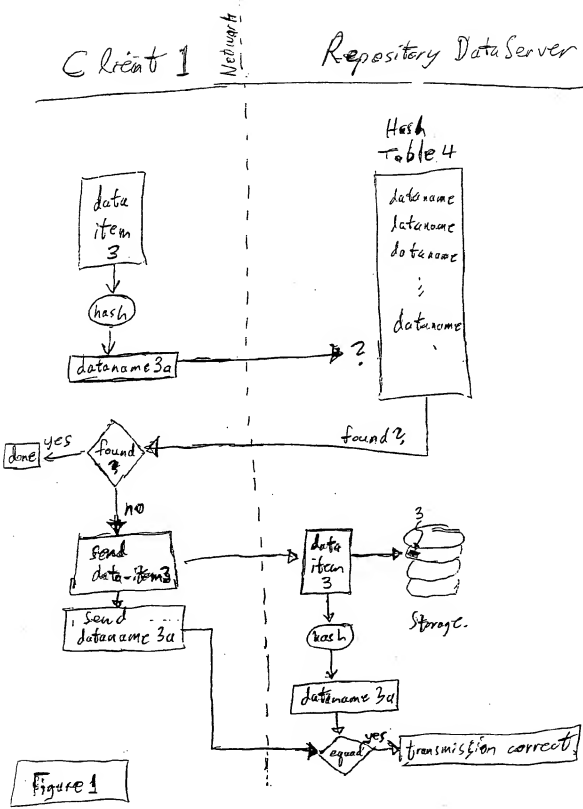

Client 1
Network
Repository Data Server 2
Hash Table 4
data item 3
hash
dataname 3a
?
dataname
dataname
dataname
dataname
found?
yes
done
found?
no
send data-item3
data item 3
3
Storage
send dataname 3a
hash
dataname 3a
equal?
yes
transmission correct

Figure 1

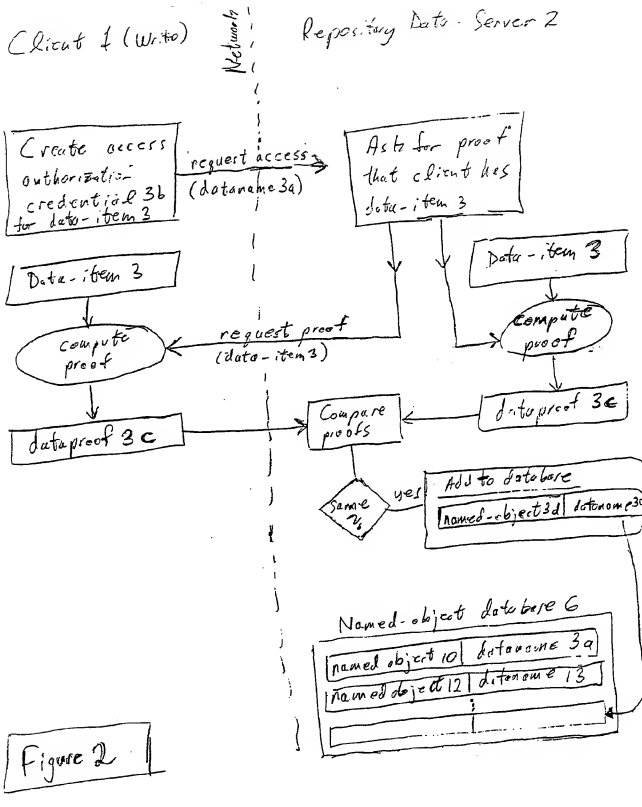

Client 1 (Write)
Network
Repository Data Server 2
Create access authorization credential 3b for data-item 3
request access (dataname 3a)
Ask for proof that client has data-item 3
Data-item 3
Data-item 3
compute proof
request proof (data-item 3)
compute proof
dataproof 3c
dataproof 3e
Compare proofs
Same ?
yes
Add to database
named-object 3d
dataname 3a
Named-object database 6
named object 10
dataname 3a
named object 12
dataname 13

Figure 2

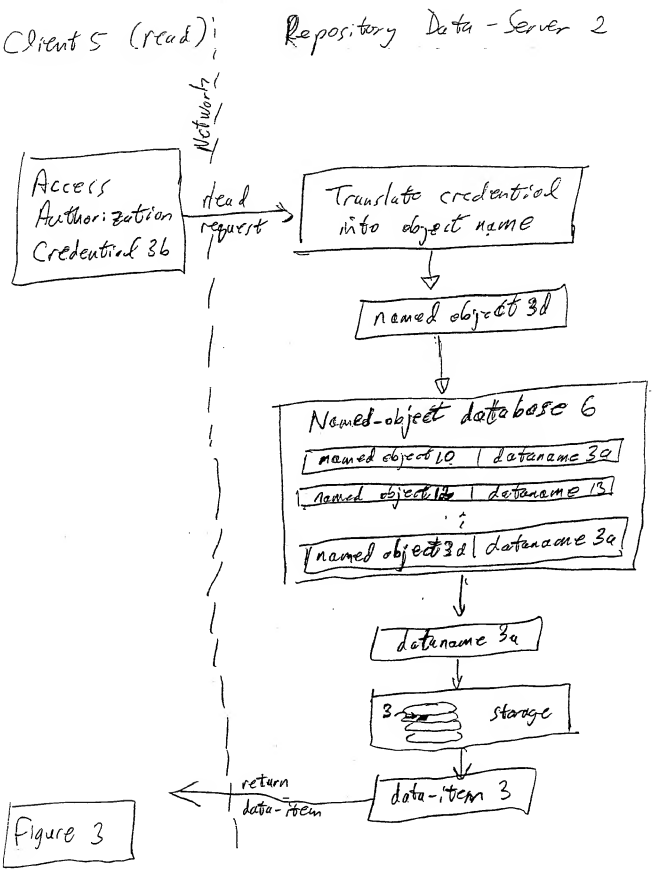

Client 5 (read)
Repository Data-Server 2
Network
Access Authorization Credential 36
Read request
Translate credential into object name
named object 3d
Named-object database 6
named object 10 | datname 3a
named object 12 | datname 13
named object 3d | datname 3a
datname 3a
3
storage
return data-item
data-item 3

Figure 3

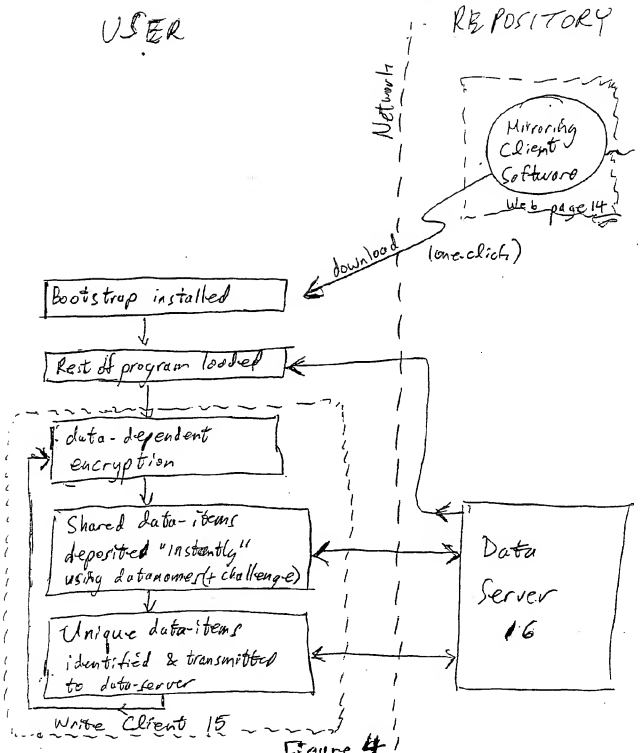

USER
REPOSITORY
Network
Mirroring Client Software
13
Web page 14
download
(one-click)
Bootstrap installed
Rest of program loaded
data-dependent encryption
Shared data-items deposited "instantly" using datanomer(+challenge)
Unique data-items identified & transmitted to data-server
Write Client 15
Data Server 16

Figure 4

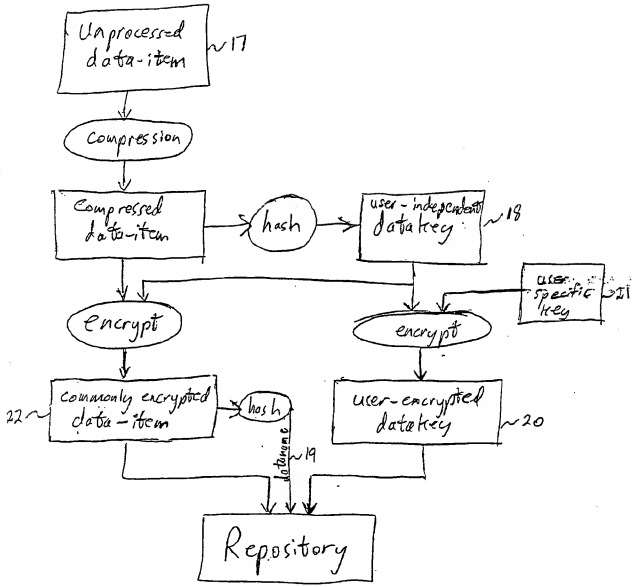

Unprocessed data-item
17
compression
Compressed data-item
hash
user-independent data key
18
user-specific key
21
encrypt
encrypt
commonly encrypted data-item
22
hash
user-encrypted data key
20
datagram
19
Repository

Figure 5

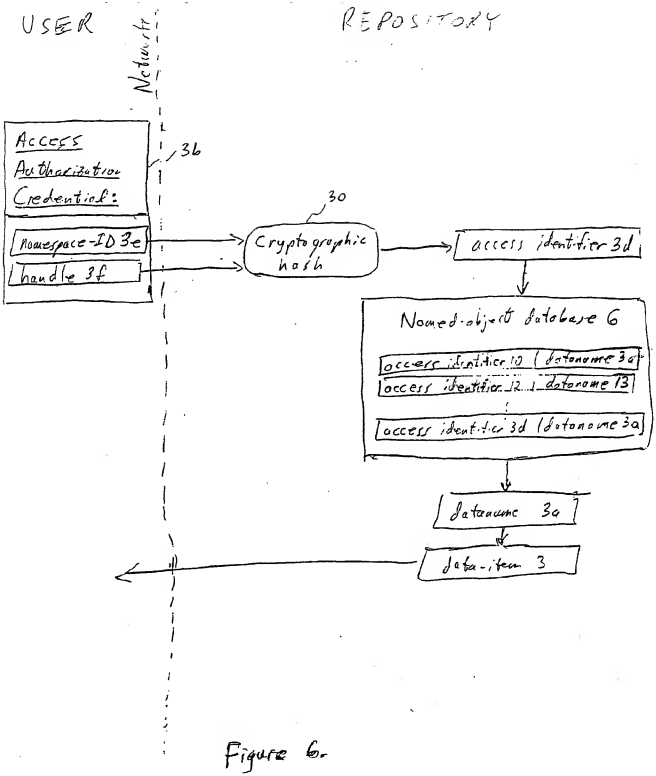

USER
Network
REPOSITORY
Access Authorization Credential:
36
Namespace-ID 3e
handle 3f
30
Cryptographic hash
access identifier 3d
Named-object database 6
access identifier 10 | datanume 3a
access identifier 12 | datanume 13
access identifier 3d | datanume 3a
datanume 3a
data-item 3

Figure 6.

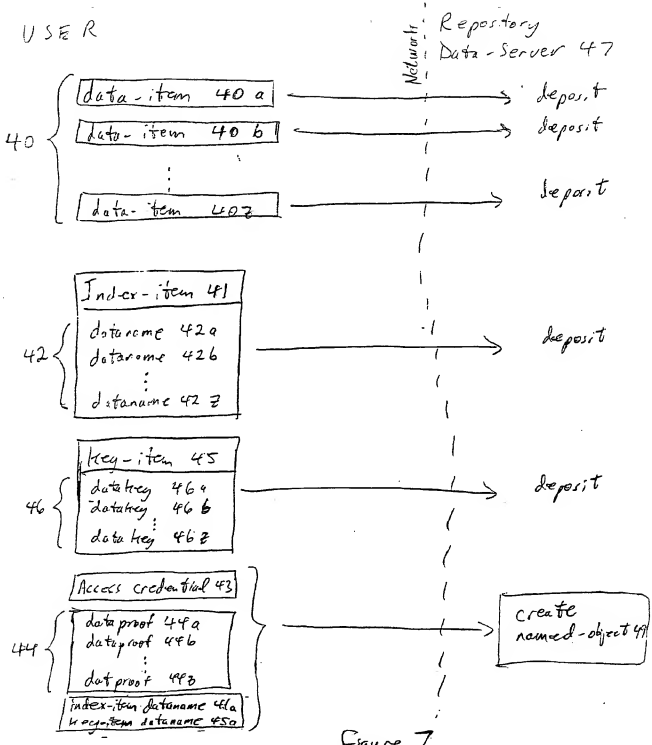

USER
Network
Repository
Data-Server 47
40
data-item 40a
data-item 40b
data-item 40z
deposit
deposit
deposit
Index-item 41
42
dataname 42a
dataname 42b
dataname 42z
deposit
key-item 45
46
datakey 46a
datakey 46b
datakey 46z
deposit
Access credential 43
44
dataproof 44a
dataproof 44b
dataproof 44z
index-item dataname 41a
key-item dataname 45a
create
named-object 49

Figure 7

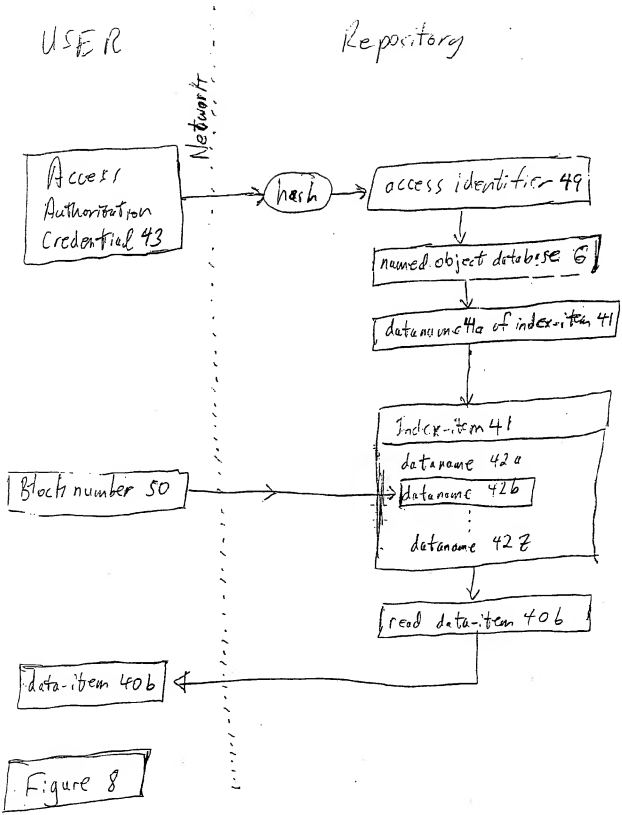

USER
Repository
Network
Access Authorization Credential 43
hash
access identifier 49
named object database 6
datanname 44a of index-item 41
Index-item 41
datanname 42a
datanname 42b
datanname 42z
Block number 50
read data-item 406
data-item 406

Figure 8

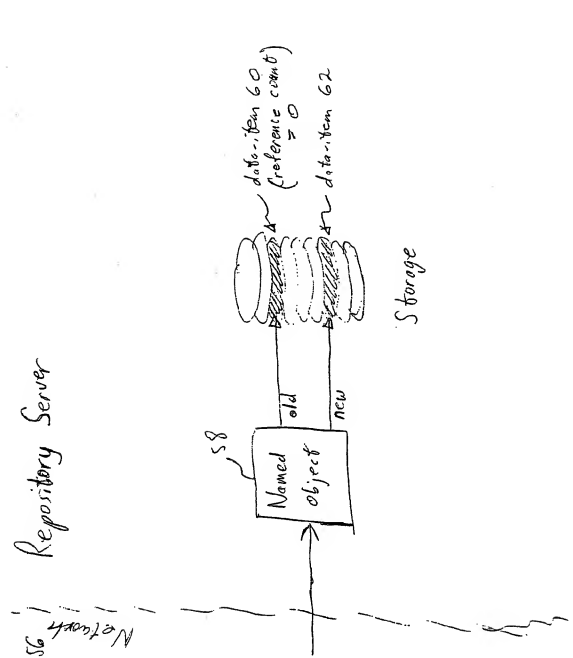

Client (write) 56
Network
Repository Server
access authorization credential
Identity Cert
58
Named objects
old
new
data item 60
(reference count > 0)
data item 62
Storage

Figure 9.

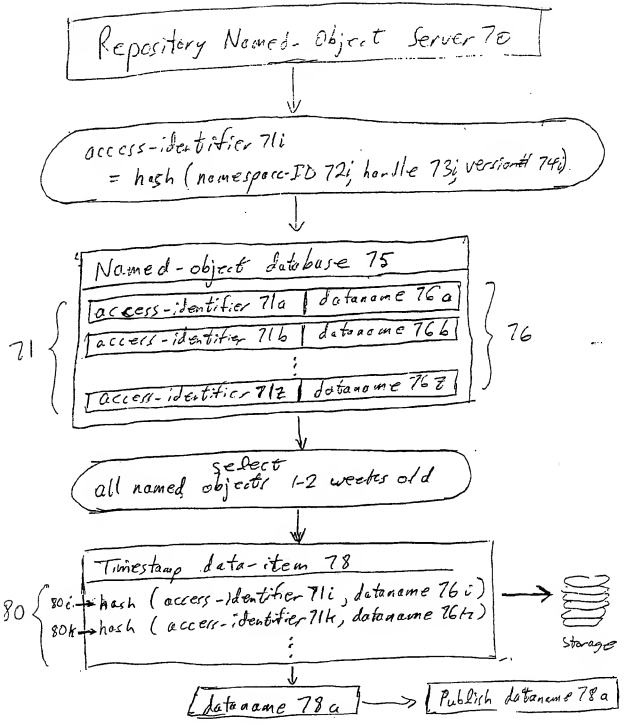
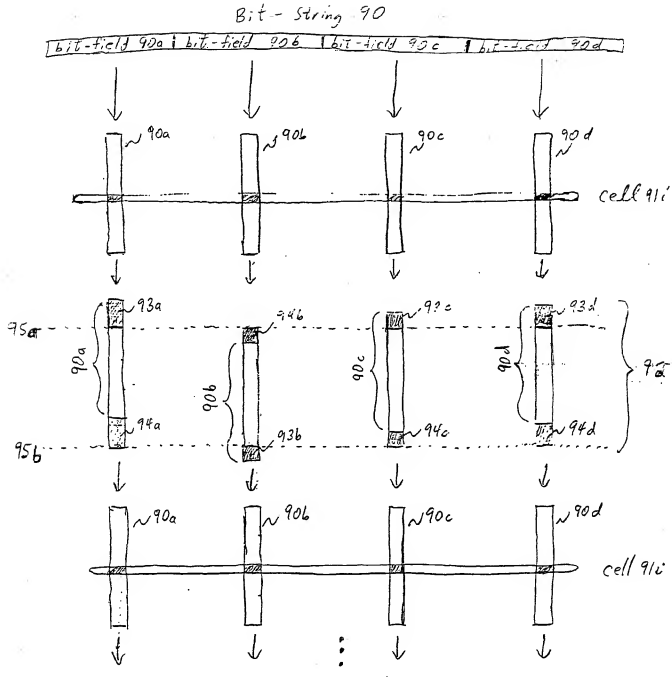

Repository Named-Object Server 70
access-identifier 71i
= hash (namespace-ID 72i, handle 73i, version# 74i)
Named-object database 75
access-identifier 71a | dataname 76a
access-identifier 71b | dataname 76b
access-identifier 71z | dataname 76z
71
76
select
all named objects 1-2 weeks old
Timestamp data-item 78
80
80i → hash (access-identifier 71i, dataname 76i)
80k → hash (access-identifier 71k, dataname 76k)
Storage
dataname 78a
Publish dataname 78a

Figure 10



Key 99: shift amount | shift amount | shift amount | shift amount | perm# ...
99a 99b 99c 99d 99e

Fig. 11

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
23 August 2001 (23.08.2001)

PCT

(10) International Publication Number
WO 01/61438 A3

(51) International Patent Classification⁷: G06F 1/00, 12/14, 11/14

(21) International Application Number: PCT/US01/05355

(22) International Filing Date: 20 February 2001 (20.02.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/183,466 18 February 2000 (18.02.2000) US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 60/183,466 (CIP)
Filed on 18 February 2000 (18.02.2000)

(71) Applicant *(for all designated States except US)*: **PERMABIT, INC.** [US/US]; 14 Portland Street, Cambridge, MA 02139 (US).

(72) Inventors; and

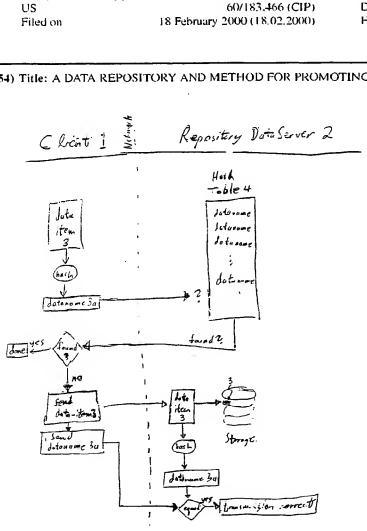
(75) Inventors/Applicants *(for US only)*: **MARGOLUS, Norman, H.** [CA/US]; 4 Aldersey Street, #24, Somerville, MA 02143 (US). **KNIGHT, Thomas, F., Jr.** [US/US]; 58 Douglas Road, Belmont, MA 02178 (US). **BOGHOSIAN, Bruce, M.** [US/US]; 6134 Lexington Ridge Road, Lexington, MA 02421 (US).

(74) Agent: **LEE, G., Roger**; Fish and Richardson P.C., 225 Franklin Street, Boston, MA 02110-2804 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR,

[Continued on next page]

(54) Title: A DATA REPOSITORY AND METHOD FOR PROMOTING NETWORK STORAGE OF DATA



(57) Abstract: In general, the invention features methods by which more than one client program connected to a network stores the same data item on a storage device of a data repository connected to the network. In one aspect, the method comprises encrypting the data item using a key derived from the content of the data item, determining a digital fingerprint of the data item, and storing the data item on the storage device at a location or locations associated with the digital fingerprint. In a second aspect, the method comprises determining a digital fingerprint of the data item, testing for whether the data item is already stored in the repository by comparing the digital fingerprint of the data item to the digital fingerprints of data items already in storage in the repository, and challenging a client that is attempting to deposit a data item already stored in the repository, to ascertain that the client has the full data item.

WO 01/61438 A3

LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) **Designated States** *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

(88) Date of publication of the international search report:
16 May 2002

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

INTERNATIONAL SEARCH REPORT

Inter. ional Application No
PCT/US 01/05355

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F1/00 G06F12/14 G06F11/14

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 0 774 715 A (STAC ELECTRONICS) 21 May 1997 (1997-05-21)	1-7,9,47
A	abstract column 7, line 37 -column 11, line 12 column 12, line 43 -column 19, line 48 column 22, line 30 -column 25, line 51 column 32, line 58 -column 38, line 59 ---	8
A	US 5 915 025 A (SAITO KAZUO ET AL) 22 June 1999 (1999-06-22) abstract column 7, line 37 -column 11, line 25 column 21, line 29 - line 56 ---	1,4-9,47
A	US 5 765 152 A (ERICKSON JOHN S) 9 June 1998 (1998-06-09) column 10, line 40 -column 13, line 25 column 23, line 9 - line 34 ---	1,4-9,47

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
13 September 2001	04 03 2002

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Jacobs, P

Form PCT/ISA/210 (second sheet) (July 1992)

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	WO 99 09480 A (TELEBACKUP SYSTEMS INC) 25 February 1999 (1999-02-25) abstract page 6, line 2 -page 21, line 18 -----	1-4,9,47

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 01/05355

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This International Search Report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the International Application that do not comply with the prescribed requirements to such an extent that no meaningful International Search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

see additional sheet

1. ☐ As all required additional search fees were timely paid by the applicant, this International Search Report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this International Search Report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☒ No required additional search fees were timely paid by the applicant. Consequently, this International Search Report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

1-9, 47

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☐ No protest accompanied the payment of additional search fees.

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

This International Searching Authority found multiple (groups of) inventions in this international application, as follows:

1. Claims: 1-9,47

   Method for secure storage of a data item on a device of a data repository connected to a network for clients programs connencted to the network comprising the step of grouping users of the method into families and the key derived from the content of the data item may be different for users in different families.

2. Claims: 10,11-32,52-67,98-113,117-119

   Method for secure storage of a data item on a device of a data repository connected to a network for clients programs connencted to the network comprising the step of associating the data item with each of a plurality of access-authorization credentials, each of which is uniquely associated with a particular user or client program.

3. Claims: 33-35,68-97,133-153

   Method for secure storage of a data item on a device of a data repository connected to a network for clients programs connencted to the network comprising a challenge step to ascertain that the client has the full data item.

4. Claims: 36-46

   Method for secure storage of a data item on a device of a data repository connected to a network for clients programs connencted to the network comprising the step of using the clients by depositors to store data items in the repository whereby at least some depositors are required to provide identification.

5. Claims: 48-51,120,121,125-132

   Method for secure storage of a data item on a device of a data repository connected to a network for clients programs connencted to the network wherein the client program using the repository is a mirroring program which determines which data item to deposit in the repository.

6. Claim : 114

   Method for secure storage of a data item on a device of a data repository connected to a network for clients programs connencted to the network comprising the steps of converting

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

widely circulated non-electronic media to a standardized electronic version, storing the standardized electronic version as a data item in the repository, promoting the availability of the standardized electronic version to users with the right to have access.

7. Claims: 115,116

Method for secure storage of a data item on a device of a data repository connected to a network for clients programs connencted to the network comprising the steps of transferring the data item which is not already in the repository over the lower speed connection from the client to the repository, the repository being connected to the network over a higher speed connection than the client, making a higher speed connection between an application server and the data repository, executing an application on the application server to process the data item stored on the data repository, returning at least some of the processed data to the client across the lower speed connection.

8. Claims: 122-124

Method for secure storage of a data item on a device of a data repository connected to a network for clients programs connencted to the network comprising the steps of dividing a bit-string into segments in which at least some bits in each segment are considered to be homologous, transforming disjoint groups of homologous bits by applying a state-permutation operation separately to each group and changing which bits are considered to be homologous and repeating the process.

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No PCT/US 01/05355

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0774715 A	21-05-1997	US 5778395 A	07-07-1998
		JP 10049416 A	20-02-1998
US 5915025 A	22-06-1999	JP 9258977 A	03-10-1997
US 5765152 A	09-06-1998	AU 7662496 A	30-04-1997
		WO 9714087 A	17-04-1997
WO 9909480 A	25-02-1999	AU 8457698 A	08-03-1999

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 August 2001 (30.08.2001)

PCT

(10) International Publication Number
WO 01/63387 A2

(51) International Patent Classification[7]: G06F 1/00

(21) International Application Number: PCT/US01/05505

(22) International Filing Date: 22 February 2001 (22.02.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/184,074	22 February 2000 (22.02.2000)	US
60/184,075	22 February 2000 (22.02.2000)	US
60/184,079	22 February 2000 (22.02.2000)	US

(71) Applicant: VISUALGOLD.COM, INC. [US/US]; Suite 203, 1011 First Street South, Hopkins, MN 55343 (US).

(72) Inventors: RICHARDS, Kenneth, W.; 7311 - 15th Avenue South, Richfield, MN 55423 (US). MURRAY, Arnold, E.; 10975 Stacy Trail, Chisago City, MN 55013 (US).

(74) Agent: BRUESS, Steven, C.; Merchant & Gould P.C., P.O. Box 2903, Minneapolis, MN 55402-0903 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AT (utility model), AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, CZ (utility model), DE, DE (utility model), DK, DK (utility model), DM, DZ, EE, EE (utility model), ES, FI, FI (utility model), GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SK (utility model), SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

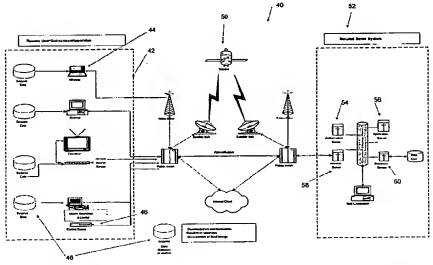
(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: SECURE DISTRIBUTING SERVICES NETWORK SYSTEM AND METHOD THEREOF



(57) Abstract: A persistent data control system and method of securely distributing data on a network includes the steps of providing an encoded file of a single file type having a plurality of file control fields, the file having at least one data type, and incorporating at least one encoded use right and/or access right into one of the control fields of the at least one data type. The persistent data control method is performed at an application level, and is capable of being embedded in an application which originates the at least one data type or called by an application. The persistent data control method further comprises the steps of decoding the plurality of file control fields including the file control fields for the encoded use right and/or access right, decoding the at least one data type in accordance with the access right, and rendering the decoded data type in accordance with the decoded use and access right.

WO 01/63387 A2

SECURE DISTRIBUTING SERVICES NETWORK SYSTEM AND METHOD THEREOF

This application is being filed as a PCT International Patent application in the name of VisualGold.com, Inc., a U.S. national corporation, designating all countries except the US, on 22 February 2001.

TECHNICAL FIELD

The present invention relates to an electronic communication network system and method thereof, and more particularly, to a secure distributing services network system and method.

BACKGROUND

In the digital communication era, security has become a large part of an electronic communication network system, particularly a network system for distributing services, such as legal services, bank transactions, etc. In many existing security systems, digital data are encoded at a transmit end and decoded at a receive end. A security system may include mechanisms for user authentication and data encryption/decryption or referred to as encoding/decoding. Also, a security system may provide public and/or private keys to authenticate a recipient and encrypt/decrypt data sent by an owner, sender, or provider of the data (hereinafter referred to as an owner of the data). However, further improvements in transferring a public and/or private key in a secure distributing services network are desired.

In addition, an owner of the data often has certain policies and/or rules that would govern and control the rendering of, access to, and/or use of that data and its lifecycle to a targeted recipient of the data. For example, an owner of the data may only want to grant a targeted recipient the ability to read the data twice in a certain time period. Further, it is desired to control and/or enforce use rights and access rights at a user application level. The existing security systems have not been designed to provide and/or enforce these and/or other policies and rules.

It is with respect to these and other considerations that the present invention has been made.

SUMMARY

In accordance with this invention, the above and other problems were solved by providing a persistent data control method of securely storing data and its use on an apparatus and/or distributing data on a network which includes the steps of: providing an encoded file of a single file type having a plurality of file control

fields, the encoded file having at least one data type; and incorporating at least one encoded use right into one of the control fields of the at least one data type.

In one embodiment of the present invention, data is encrypted and formatted in a single file type. The encoded file includes a plurality of file control fields. At least one of the fields incorporates the persistent data control policy that controls use rights and/or access rights of a recipient. The persistent data control policy is granted by an owner.

In one embodiment of the present invention, data is encrypted and formatted in a database structure. The database structure includes a plurality of database structure control fields. At least one of the control fields incorporates the persistent database structure control policy that controls use and/or access rights of a recipient. The persistent database structure control policy is granted by the owner of the database.

Still in one embodiment, the data type may include, but not limited to, digital files, and a database structure or its elements including static image, video, text, markup language (e.g. HTML), etc.

Further in one embodiment of the present invention, the secure embedded database includes a plurality of fields which define arbitrary descriptions, file size(s), file type(s), etc.

Additionally in one embodiment of the present invention, the file(s) and their descriptions can be queried and returned independently by supplying values for a search keyword that is defined in the descriptions, without decoding the entire encoded data in accordance with encoded user access rights and use rights.

In one embodiment of the present invention, the persistent data control method is performed at an application level.

Still in one embodiment, the persistent data control method is capable of being embedded in an application which originates the at least one data type. Alternatively, the persistent data control method is called by an application.

Yet in one embodiment, the data may be encoded in a memory buffer and decoded from a memory buffer (i.e. buffer-to-buffer), or encoded in a file and decoded from a memory buffer (i.e. file-to-buffer), or encoded in a memory buffer and decoded from a file (i.e. buffer-to-file), or encoded in a file and decoded from a file (i.e. file-to-file).

Further in one embodiment, the persistent data control method further comprises the step of incorporating multiple encoded use rights into the control fields of the at least one data type.

Additionally in one embodiment, the persistent data control method further comprises the step of incorporating at least one encoded access right into one of the control fields of the at least one data type.

Yet in one embodiment of the present invention, the encoded use right is encoded with the at least one data type. Alternatively, the encoded use right is encoded independently from the at least one data type.

Still in one embodiment, the persistent data control method further comprises the steps of: decoding the plurality of file control fields including a file control field for the at least one encoded use right; decoding the at least one data type; and rendering the decoded data type in accordance with the decoded use right.

In another embodiment of the present invention, the persistent data control method further comprises the steps of: decoding the plurality of file control fields including a file control field for the at least one encoded use right; decoding the plurality of the file control fields including a file control field for the at least one encoded access right; decoding the at least one data type in accordance with the decoded access right; and rendering the decoded data type in accordance with the decoded use right.

The present invention also includes a persistent data control system for securely distributing data on a network. The persistent data control system includes: an encoded file of a single file type having a plurality of file control fields, the file having at least one data type; and means for incorporating at least one encoded use right into one of the control fields of the at least one data type.

Still in one embodiment, the persistent data control system includes: a mechanism for authenticating a user; a mechanism for encrypting/decrypting data; and a mechanism for generating a dynamic key on a secure server and transferring the dynamic key to a recipient device. In one embodiment, the dynamic key physically resides in a memory for the term of a communication session, the time defined by the owner of the data, or the life of data being rendered. The dynamic key is generated dynamically for a session and/or specific data.

The present invention further includes a method of authenticating the encoded data. The method may generate a single file type that is verifiable so as to prevent attacks and spoofing of the encoded data. For example, the single encoded file type may be checked at a firewall or a proxy to validate the data before allowing it to enter into a system, and decoded to prevent unauthorized access or attacks on the system.

The present invention also relates to a method of distributing data on a secure network system. The method includes the steps of: authenticating a user, encrypting of data with a security key, generating a dynamic key on a secure server

and transferring the dynamic key to a recipient device, and decrypting the data by the security key based on the dynamic key transferred with the data or transferred independently of the data.

In one aspect of the present invention, the step of generating the dynamic key on the secure server and transferring the dynamic key to the recipient device includes generating the key dynamically for a session and/or specific data. In one embodiment, the dynamic key physically resides in a memory for the term of a communication session, the time defined by the owner of the data, or the life of data being rendered.

Unlike conventional encryption methodologies that apply encryption after or decode before the data generating or rendering application, the method in accordance with the present invention may be incorporated as part of the data generating and rendering application to facilitate the process and further insure the security of the information. For example, the method according to the present invention is a part of video codec and encodes each frame or a critical component of each frame while being assembled as a video. Accordingly, the present invention allows to securely and efficiently distribute digital data streams, for example, video or voice data streaming while such streams are being generated. In addition, the present invention allows for securely and efficiently re-applying an encoding process to the data multiple times to increase the degree of security.

The method according to the present invention also allows an owner of the data to define rules for rendering, accessing, and using the encoded data. Such rules can be a part of an encoding scheme. The rules are enforced when a recipient decodes the data.

For a better understanding of the invention reference should be made to the drawings which form a further part hereof, and to accompanying descriptive matter, in which there are illustrated and described specific examples in accordance with the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

Fig. 1 is a functional block diagram illustrating exemplary electronic communication methodologies for a remote authorization process.

Fig. 2 is a functional block diagram illustrating exemplary secured data distribution methodologies for a remote authorization process.

Fig. 3 is a flow diagram of one embodiment illustrating a remote authorization to render data in accordance with the principles of the present invention.

Fig. 4 is a schematic view of an exemplary composite file having one or more data type components and control components of a persistent data control system in accordance with the principles of the present invention.

Fig. 5 is a schematic view of exemplary types of an encrypted file as defined by a header of a secured embedded database of the persistent data control system in accordance with the principles of the present invention.

Fig. 6 is a functional block diagram illustrating exemplary method of encoding secured data in accordance with the principles of the present invention.

Fig. 7 is a functional block diagram illustrating exemplary method of decoding secured data in accordance with the principles of the present invention.

Fig. 8 is a schematic view of one embodiment of secured embedded database and search engine in accordance with the principles of the present invention.

Figs. 9A-9B are flow diagrams of one embodiment illustrating a method of establishing a secured session with a registered user in accordance with the principles of the present invention.

Figs. 10A-10F are functional block diagrams of various embodiments illustrating a method of registering and establishing a secured session with a new registered user in accordance with the principles of the present invention.

Figs. 11A-11B are functional block diagrams of various embodiments illustrating a method of requesting for specific content or data key and rendering in accordance with the principles of the present invention.

Figs. 12A-12D are functional block diagrams of various embodiments illustrating a method of establishing a secured session with a registered user in accordance with the principles of the present invention.

DETAILED DESCRIPTION OF ILLUSTRATED EMBODIMENTS

In the following description of the illustrated embodiments, reference is made to the accompanying drawings that form a part hereof, and in which is shown by way of illustration several embodiments in which the invention may practiced. It is to be understood that other embodiments may be utilized as structural changes may be made without departing from the spirit and scope of the present invention.

The present invention provides a persistent data control method of securely distributing data on a network which includes the steps of: providing an

encoded file of a single file type having a plurality of file control fields, the encoded file having at least one data type; and incorporating at least one encoded use right into one of the control fields of the at least one data type.

Data is encrypted and formatted in a single file type. The encoded file includes a plurality of file control fields. At least one of the fields incorporates the persistent data control policy that controls use rights and/or access rights of a recipient. The persistent data control policy is granted by an owner of the data. Alternatively, data is encrypted and formatted in a database structure. The database structure includes a plurality of database structure control fields. At least one of the control fields incorporates the persistent database structure control policy that controls use and/or access rights of a recipient. The persistent database structure control policy is granted by the owner of the database.

The data type may include, but not limited to, digital files, and a database structure or its elements including static image, video, text, markup language (e.g. HTML), etc. The secure embedded database includes a plurality of fields which define arbitrary descriptions, file size(s), file type(s), etc. The file(s) and their descriptions can be queried and returned independently by supplying values for a search keyword that is defined in the descriptions, without decoding the entire encoded data in accordance with encoded user access rights and use rights.

The persistent data control method of the present invention can be performed at an application level. The method is capable of being embedded in an application which originates the at least one data type or being called by an application.

The present invention also provides a persistent data control system and method thereof. The persistent data control system includes a mechanism for authenticating a user, a mechanism for encrypting/decrypting data, a mechanism for generating a dynamic key on a secure server and transferring the dynamic key to a recipient device, and a mechanism for authenticating the encrypted data.

It is appreciated that various standards of user authentication can be used within the scope of the present invention. Examples of user authentication methods are as follows:

1) Basic authentication methods:

i) Challenge handshake authentication protocol (CHAP) response – encrypted user name and password transfer;

ii) Basic or PAP (Password authentication protocol) clear text transfer authentication; or

iii) 2-factor authentication – server to client and client to server when coming over.

2) Certificate of Authority (CA) - where a third party provides user authentication to the server; or

3) Digital Signatures - where an owner signs its identification in a digital format.

The persistent data control system in accordance with the present invention may incorporate the above authentication standards to authenticate a user to a server or between any two users, devices, or applications. Once a user is authenticated, the persistent data control process uses an encryption schema for data communications to transfer a dynamic key generated on a secure server to a persistent data control application on a recipient device.

It is also appreciated that various standards of encryption/decryption methods can be used within the scope of the present invention. Standard encryption/decryption methods are hardware and software solutions that encrypt/decrypt based on a defined protocol between the two communicating devices and exchange of keys. The persistent data control system in accordance with the present invention may use or incorporate the same encryption/decryption schema as that is used for communication between devices, for example the Data Encryption Standard (DES) or Blowfish (A 64-bit block symmetric cipher consisting of key expansion and data encryption), etc. In one embodiment of the persistent data control system of the present invention, the data is encrypted at an application level using the same or another cipher and then be encrypted by a protocol used by a network which may be arbitrated or not.

A dynamic key used in connection with the persistent data control system of the present invention is a key that is not physically stored on a device but resides only in a memory for the term of a session, time defined by an arbitrating device such as the server, or for the life of a data being rendered. The key is generated dynamically for a specific session and/or specific data. A dynamic key can be transferred via a standard encryption protocol that is used by a network for establishing the dynamic key for a session as shown in Figs. 10A-10F, 11A-11B, and 12A-12D. Alternatively, a dynamic key can be transferred through the use of headers as shown in Figs. 4 & 5. The dynamic key is changed on the fly for each session or for a specific data. The dynamic key preferably resides in a memory for the term of a communication session, the time defined by the owner of the data, or the life of a data being rendered.

In addition, the persistent data control system of the present invention controls the access of the encrypted data based on a set of rules or policies and enforces the rules or policies at an application level upon rendering the data at a recipient end.

The data is preferably encrypted and formatted in a file type format. The file includes a designated portion, for example, a header, which has a plurality of fields. At least one of the fields defines a rule and/or policy that controls use rights and access rights of a recipient. The use rights and access rights are granted by an owner of the data.

The persistent data control system includes a secure embedded database and a search engine. The data may include digital files, their descriptions, user rights of access, rendering, and use. The data are stored in a secure searchable structure. The secure embedded database includes a plurality of fields that define arbitrary descriptions, file size(s), file type(s), and an arbitrary number of files associated with the descriptions. Further, the file(s) and their descriptions are preferably queried and returned independently by supplying values for a search keyword that is defined in the descriptions.

Fig. 1 illustrates exemplary electronic communication methodologies of a persistent data control system 40 for a remote authorization process for accessing and using secured data 48. A remote user/subscriber/apparatus 42 may be any one of a wireless electronic device, a desktop computer, a television, a remote access device, a mobile device, a laptop, another server, or others that would become apparent to one skilled in the art. The remote apparatus 42, such as a desktop or laptop device, may have the communications and data control application process or device incorporated therein for providing encryption/decryption access and control of the received and sent secured data or database. As shown, the remote apparatus 42, such as the television, the desktop computer, the mobile device, and the laptop, may be connected to a communications and/or control device 46 incorporating the data control application process for providing and controlling encryption/decryption and control of the received and sent secured data or database.

In Fig. 1, the remote user/subscriber/apparatus 42 is in communication with a secured data 48 or has received a secured data 48 that is either downloaded via communication to the apparatus 42 or is available on removable or fixed storage media. The secured data 48 may be transferred from an owner of the secured data, through various communications channels 50, such as radio towers, public switch networks, satellite dishes, optical fiber, copper wire, the Internet, etc., to a recipient apparatus 42 or secured server system 52. At the secured server system 52, an authorization server 54, an application server 56, an Internet server 58, a database server 60 are interconnected through a network, e.g. the Ethernet, to provide services and exchange of the secured data. The secured server system 52 generate all dynamic keys for an encoded session as well as the secured data 48, and provide the keys and the data via the communications channels 50 to the remote

apparatus 42 incorporating the controls 46 and application 44 for decoding and enforcing of the policies and rules associated with the secured data or database 48. The remote user/apparatus 42 may further encode secure data or changes to the secure database 48 and send such encoded data to the secure server system 52 for rendering the data or database update, or to another remote user/apparatus 42 for rendering in accordance with the rules and policies incorporated therein.

Fig. 2 illustrates exemplary secured data distribution methodologies. Secured data 62 is downloaded from a remote site 64 to a secured server system 66 via communication media 68, such as the Internet, then to a recipient 67 via the media 68. Alternatively, the secured data 62 is stored on removable storage media 70 and delivered manually via a postal service 72 or courier 74 to the recipient 67.

Fig. 3 is a flow diagram of one embodiment illustrating a remote authorization process 76 to render data in accordance with the principles of the present invention. The process 76 starts with an operation 78 of establishing a connection with a server. Then, a request for subscription and access by a user/apparatus to the persistent data control system is sent to the server in an operation 80 along with a subscriber ID in an operation 82. Next, a connection is established with the server in an operation 84, and a new subscription and data access request is processed in an operation 86. Then, the subscriber ID is processed in an operation 88. If the subscriber ID is determined in an operation 90 to be invalid, i.e. the "no" path, an ID error is indicated in an operation 92 that terminates the process 76. If the subscriber ID is determined in the operation 90 to be valid, i.e. the "yes" path, then a secured session is built in an operation 94. Then, a request for rendering of the secured data is made in an operation 96. Next, access and user rights policy of a recipient is processed in an operation 98.

The process 76 may determine whether a payment is required for rendering the secured data in an operation 100. If no payment is required, i.e. the "no" path, an authorization key and user access and use rights are given to the recipient in an operation 102, and the authorization key and user access and use rights are ~~is~~ used to render the secured data to the recipient in an operation 104..

If a payment is required from the operation 100, i.e. the "yes" path, a request for payment is sent to the recipient in an operation 106. The recipient may respond by sending a payment method in an operation 108. Then, the payment is processed in an operation 110, and the authorization key and user access and use rights are ~~is~~ sent to the recipient in the operation 102. Next, the authorization key and user access and use rights are used to process and render the secured data. Then, the process 76 is terminated.

Fig. 4 is a schematic view of an exemplary composite file having one or more data type components and control components of a persistent data control system in accordance with the principles of the present invention. Fig. 4 illustrates control information including the control components, such as header elements, policy elements, and access map elements, etc. Fig. 4 also illustrates data type information including data type components, such as database elements, data elements, etc. The data or datum is encrypted in a file format or type that preferably includes a header component 112, a policy component 114, a database component 116, an access map component 118, and a data component 120.

As shown in Fig. 4, the header component 112 includes elements such as a header length, type, policy elements, composite hash element of the encoded data, database pointer, database length, access map pointer, access map length, one or more file pointers, file name(s), file length, encryption key (E key), etc. A further detailed description of the elements of the header component 112 is shown in a box 112'. A header length is varied depending on different types of persistent data control methods. The policy component 114 is incorporated into one of the elements of the header component 112. Also, pointers to various other components, such as a descriptive database composed of discrete elements, access rights map, first encrypted file data, and possibly next encrypted file data, are incorporated into elements of the header component 112. In addition, an encryption/security key for accessing the database and other encrypted file data is incorporated into one of the elements of the header component 112. It is appreciated that the elements in the header component 112 can be embedded in anywhere within the encoded composite file and data type files without departing from the present invention, for example, a footer, etc. For simplicity and illustration, a header component is hereinafter described as an example.

The policy component 114 includes elements that define recipient's access rights to the data, such as the rights to "read/write", "save encoded", "save open", "no save", "server keyed", "render 1", "render 2", "Age 1", "Age 2", and "Use", etc. A further detailed description of the elements of the policy component 114 is shown in a box 114'. The "read/write" element indicates that full rights are granted to a recipient of the data. The "save encoded" element allows the recipient to save the data on its system only as an encrypted file. The "save open" element allows the recipient to save the data on its system in an original open format of the data. The "no save" element only allows the data to reside in a memory and to be erased upon closing of the data file by the recipient, upon aging after a certain period of time, or a pre-defined user element, etc. The "server keyed" element allows the recipient to work in conjunction with "save encoded" element. The "server keyed"

element requires the recipient to authenticate itself to the server and request opening of a file. A required key will be provided by the secure server. The "render 1" element and "render 2" element allow the recipient to render the data on different ports, such as a CRT or a printer, etc. The "age 1" element defines a specific date that the recipient needs to render the data so as to prevent spoofing. The "age 2" element provides a specific time and date that an encrypted file will be erased from the system. The "age 1" element and "age 2" element may work in conjunction with the "server keyed" element. The "use" element defines the number of times that the data may be accessed or used. The "use" element may work in conjunction with the other policy elements.

As shown in Fig. 4, the exemplary database component 116 includes elements "Key 1", "E1", "K2", "E4", "E5". A further detailed description of the elements of the database component 116 is shown in a box 116'. The database elements can be defined by an owner or can be a representative of an existing database that may be an encoded copy of a query, a record of a database, or a composite file, etc. Searches of the database are performed in such a manner that it does not require opening of the encoded file or database and limit access to its elements according to the map access rights elements 118 and limit the rendering in accordance with the policy components 114. Also, search keys may be a part of an encrypted database whereby an index table can be rebuilt to reduce loss of database integrity. In addition, the policy component 114 and the access map component 118 may work in conjunction with the database component 116 to enforce the use and access rights granularity.

In Fig. 4, the exemplary access map component 118 includes elements "Group(x)", "Rules/Rights", "K_{1-n} element read index", "E_{1-n} element write index". A further detailed description of the elements of the access map component 118 is shown in a box 118'. The access map elements define access to individual data elements by user group, and the type of rights granted, e.g. read only, write only, read/write, etc.

The exemplary data component 120 includes one or more data elements. A further detailed description of the data elements is shown in a box 120'. One or more data elements may exist depending on a header type. Digital data may be of any type and length. Data may also be streamed from one source to another, encrypted from file to buffer, buffer to buffer, buffer to file, or file to file.

It is appreciated that other components may be included in a database file within the scope of the present invention. Also, it is appreciated that other elements may be included in each of the components without departing from the scope of the present invention.

It is also appreciated that since all encoded header data, database, and any other data are encoded as a single data file or stream being singular in type, the data may be checked by the application before opening via the various embedded hash elements. Accordingly, the security and integrity of the data is further maintained, firewall requirements are simplified, and the potential of firewall penetration is reduced.

Fig. 5 is a schematic view of different types of an encrypted file as defined by a header of a secured embedded database of the persistent data control system in accordance with the principles of the present invention. In type 1, a file 122 has a header element without other elements. The type 1 file 122 is a key application for a request from the user/device/application for a data encryption key and its transfer from the secure server. In type 2, a file 124 includes the header element with the policy element and data element. The policy element defines the policy for delivered and embedded data. In type 3, a file 126 includes the header element with the policy element, database element, and data element. The policy element defines the policy for delivered database and embedded data. In type 4, a file 128 includes the header element with the policy element, access element, and database element. The policy element defines the policy for delivered database. In type 5, a file 130 includes the header element with the policy element, access element, database element, and data element. The policy element defines the policy for accessed, delivered, and embedded data. In type 6, a file 132 includes the header element with the policy element, access element, database element, data element, another header element with a policy element and data element. The policy elements define the policies for delivered database and multiple embedded data. In type 7, a file 134 includes the header element with the policy element, access element, database element, data element, another header element with a policy element, access element, database element, and data element. The policy elements define the policies for multiple accessed, delivered, and embedded data.

Fig. 6 is a functional block diagram of one embodiment of a method 136 of encoding secured data component in accordance with the principles of the present invention. Illustrated are the interface components, the secure software or logic components, and the secured data output. An owner of the data instantiates a request for an encoding process in block 138. Then, encoding parameters in block 140 which are input via data I/O format and level logic are used to set logic flow for setting up the encoding process in block 142. Next, the process 136 determines whether a file is a single file or multiple files in block 144. The determination may be made based on a data path or data origin. Next, the process 136 generates a file header based on the rights and rules defined by the owner in block 146. Then, the

encoder process 136 generates a master seed based on a time stamp, a license key, an apparatus key, and a dynamic key in block 148. Next, an encoding template is generated in block 150 based on the master seed and key set for the encoding of the data components and of the final composite file. Then, the input data is encoded according to the encoding template in block 152. Finally, the encoded data is outputted to a file or a buffer that include both the encoded data and the header in block 154.

Fig. 7 is a functional block diagram of one embodiment of a method 156 of decoding a secured file in accordance with the present invention. Illustrated are the interface, the secure software or logic components, and the secured data output components. A recipient of the data instantiates a request for a decoding process in block 158. The data in the received file or buffer is decoded into a header component and a data component in block 160. Then, the process 156 reads the header in block 162 to determine the file destination and output format. Next, the process sets up a decode level and logic flow in block 164. Then, a master seed is generated in block 166 that determines a license key, an apparatus key, and a dynamic key. Next, a decoding template is generated in block 168 based on the master seed and the key set for decoding the data components and the final composite file. Further, the header is decoded to determine the policies and rules for the recipient's use rights of the data in block 170. Finally, the data is decoded based on the user rights in block 172.

Fig. 8 is a schematic view of one embodiment of secured embedded database and search engine 280 in accordance with the principles of the present invention. Illustrated are interfaces, a secure database record generation process 282, a secured data or database output process 296, a search engine and secure query output process 304. The secure database record generation process 282 is initiated upon recipient of a data class definition 284, a database element structure in block 286, a user data access group definition in block 290, and data elements of the record in block 294. The received information may be provided from existing databases and security components of the system or via a custom interface where they may be entered as required. The data class in block 284 is used by the record structure definition in block 286 to organize the data elements for building of an encoded database in block 288. Furthermore, the defined data class in block 284 is used to generate a unique file folder 298 of the secured data or database output process 296 for all records generated using a given data structure. The data security schema in block 290 is mapped to the encoded database built in block 288 by block 292 to define the user group access rights to individual data elements as defined by the owner of the database and presented to the appropriate interface. The output of the

encoded database block 288 generates a database key index file 300 for later queries by a search engine. The database key index file 300 may be encoded. Each independent data record using the mapped database structure generated by block 292 may be entered and mapped into a database according to block 294. The mapped data from block 294 and any other input data is encrypted according to the secure encode components of the process 136 and output to the appropriate class folder 298 for the defined database structure. The mapped data record in block 294 updates the database key index file with each new set of search keys and indexes for each new data record entered using the same structure.

The secured data or database output process 296 generates a unique class folder, e.g. the class folder 298, for each unique database structure generated from the build database block 288 for a set of data records. A unique key index file, e.g. the index file 300, is created for each unique structure created in block 288 and is updated with the keys and index data for each record having the same unique class and database structure. An encoded database and data record 302 is generated by the secure encoded components in the process 136 and contains all user rights to which the user has access rights as defined and mapped by block 292.

The secure query output process 304 is initiated by a user requesting a specific data by a user having a search engine 306 and a secure encode/decode application software. The search engine 306 receives query information in block 308 composed of the keys, path and output form for the queried data as well as the data class if required that is provided a the class query in block 310. The search engine 306 opens the appropriate class folder 298 or searches all class folders having the same key for records that meet the query from block 308. Each encoded database record file 302 that matches the key is presented to the secure decode components in the process 156. The secure decode components decode only those elements the user has rights to based upon the user's group definition and the encoded rights to the individual data elements and embedded data. The secure decode components in the process 156 provide the resultant decoded data to data formatting and secure rendering and viewing application in block 312.

Turning now to Figs. 9A-9B, one embodiment of a flow diagram for establishing a secured session with a registered user in accordance with the principles of the present invention.

Generally, a secured session can be established in an environment comprising the following components: (1) an Internet browser or application program that includes a persistent data control application for securely encoding and decoding digital data on a networked or remote apparatus; (2) one or more servers or another remote or networked computer which includes control, communication and

application programs, the data, and the persistent data control application for securely encoding and decoding data; and (3) a communications medium, which may be public or private and which may be wireless, satellite, landline or a local network, over which the server and a remote apparatus establishes a communication link.

For purposes of describing and illustrating the process of establishing a secured session, the following description of the illustrated embodiments utilizes the Internet as an example of a relevant communications medium. However, it will be appreciated by those skilled in the art that the present invention is not limited to the use of the Internet as any suitable computer network may be substituted without departing from the spirit and scope on the present invention.

When the Internet is the communications medium, the application to establish a secured session resides on an Internet server, or another server, which will be referred to as a secure server. The secure server makes its resources available to an Internet server. Throughout the remainder of this description of the various embodiments of the present invention, the server on which the persistent data control application resides will be referred to as the secure server.

A browser application residing on the remote apparatus has access to the persistent data control application. A secured session is configurable to meet a security policy of the data owner and can be customized to control the rendering, access and use of the secured data residing on the remote apparatus according to a set of rules defined by the owner.

The implementation of a secured session may involve the utilization of multiple encryption keys. An example of utilizing five encryption keys is presented below:

1. The first key is a fixed internal or private key accessible only by an internal code used to open a header of the encoded data.

2. The second key is a dynamic public key that may be changed with each new session or block of encoded secured information sent by the secure server as a part of a secured session.

3. The third key is a license number of the persistent data control application installed on a remote apparatus. This private, unique key is a part of a registry database and a part of the persistent data control application on the secure server, and is accessed by a hashed unique browser or user identifier associated with the persistent data control application installed on the remote apparatus. The unique identifier associated with a unique license number is embedded in the persistent data control application installed on the remote apparatus. The unique identifier is encoded and passed to the secure server. As such, the identifier may be is known

prior to initiating the first secured session and is therefore not transmitted across the Internet.

4. The fourth key is a unique identification number of the remote apparatus on which the persistent data control application is installed. This is also a private, unique key that is encoded using the persistent data control application and transmitted over the Internet one time only as a part of the initial persistent data control application registration. The secure server adds the fourth key to its persistent data control application registry database and associates the fourth key with the corresponding license number of the persistent data control application installed on the remote apparatus and the unique browser identifier. Before decoding any secured block of information that has been received by a remote apparatus, the persistent data control application installed on that remote apparatus retrieves the unique machine identifier, e.g., manufacturer's serial number, of that apparatus and uses it as one of the decode/encode keys. If the decode is successful, the apparatus has been validated.

Furthermore, the persistent data control application passes the unique machine identifier to the secure server where the machine identifier is in the registry database and is used as one of the encode/decode keys for that specific remote apparatus. This prevents any attempts of unauthorized decoding of secured information on any other apparatus. In addition, the persistent data control application will inform the secure server that an unauthorized attempt has been made to decode secured information so that an appropriate action can be taken. Such action may comprise erasing the secured data from the remote apparatus or disabling the apparatus from obtaining a secured session by posting status in a secure server registry database.

5. The fifth key is an optional key that can be implemented at a host Web site according to the requirements of the data owner. As an example, a user password or a digital signature or a server controlled key could be used separately from or in tandem with, the authentication server described below.

As described herein, a secured session is built in several stages. Each successive handshake between a remote apparatus and the secure server delivers the session to a more secure level until ultimately all data is encoded using a single set of keys that lock the remote apparatus, the user, and the secure server into the secured session. These same circumstances apply for all transmissions originating either at the remote apparatus or a server.

Two forms of a secured session may be built. The secured session may be initiated through either a public Web site and/or a private Internet network. Furthermore, the secured data can be rendered, accessed or used by a remote

apparatus upon establishing a communication session with a control apparatus that provides information and key(s) to unlock the secured data for rendering, accessing, or using by the remote apparatus. These three components will be described below.

I. **Secured Session Form 1: Public Web Site**

As an example, the first form of a secured session allows a session to be initiated through a public Web site. All other services that are provided by the Web site to the public are also available, thus requiring only one hosted Web site. However, the secured data is accessible only to those remote browsers which have the persistent data control application and which are subscribers to the secured services of that Web site.

A connected browser to which the persistent data control application has been integrated initiates a first-time secured session with a secure server. The persistent data control application encodes a block of data having the following three unique components: (1) a unique encoded header; (2) the encoded data; and (3) a unique persistent data control application file extension. The header and the file extension are specific to the persistent data control application. A unique, dynamic public key used to encode the unique identifier of the browser's persistent data control application is placed in the encoded header. The secure session having a requirement for user authentication is initiated upon such authentication using existing standards for authentication, such as a digital signature method or a public/private key exchange. A dynamic key generated by the secure server may then be securely transmitted to the browser secure application utilizing the standard digital signature exchange or public/private key encryption scheme. Such dynamic key is retained in static memory for a maximum period of the duration of the session and is not stored on a permanent storage medium. A unique identifier of the browser's persistent data control application is encoded, which will be the first of the three keys required to fulfill a secured session between the secure server and the browser. The browser sends this unique encoded block of data via the Internet to the secure server, where the file extension and header type is recognized and passed to that server's persistent data control application for decoding. The secure server uses the unique browser identifier to look up the associated unique key located within the persistent data control application registry database. This first unique key is the license number for the connected browser's persistent data control application.

The secure server begins building a secured session for the browser that will exist until the secured session is terminated. The secure session having a requirement for user authentication is initiated upon such authentication using existing standards for authentication such as a digital signature method or a

public/private key exchange. A dynamic key generated by the secure server may then be securely transmitted to the browser secure application utilizing the standard digital signature exchange or public/private key encryption scheme. Such dynamic key is retained in static memory for a maximum period of the duration of the session and is not stored on a permanent storage medium. The persistent data control application creates a key set including the public key combined with the first unique key and the dynamic key when specified by the system. The secure server encodes on that key set a request for the second unique private key. The browser decodes the request on the key set and responds by retrieving the unique machine identifier of the remote apparatus on which the persistent data control application is installed and from which the browser is operating. The browser then encodes the second unique key, e.g. the unique machine identifier, which, in combination with the previous key set, forms a final key set for all future encoding and decoding. This final key set and the dynamic key are used by the secure server and the browser for all transmissions during this secured session and for all future secured sessions between this browser/remote apparatus and this secure server.

If the licensed persistent data control application associated with that specific remote apparatus were to be re-installed on another apparatus, the secure server detects an error. In other words, the link between the persistent data control application license and the remote apparatus ID forever associates or locks that first secured session and each subsequent secured sessions initiated by that licensed persistent data control application user to the specific remote apparatus from which the first security session was initiated.

The secure server finalizes the building of the secured session by registering the second unique key in the registry database, and encoding status of the secured session established and sending it to the remote browser. All future data will be encoded and decoded. Once the secured session is established, the HTML, frames, JAVA applets and tables, only the data associated with the HTML page, or any other data formatted for a specific application using a secured session is secured, depending upon how and where the secured session is installed on the secure server and on the remote apparatus. All subsequent connections with the secure server by a remote browser that is registered require the user authentication process, the generation and passing of the dynamic key to the browser and the browser returning the encoded unique identifier to establish a secured session.

Furthermore, an alternative method if the Web site requires user authentication, the secure server will, before establishing the secured session, present an encoded request to the browser for a user password or digital signature. The browser will respond to the request by submitting the user's password and/or a

digital signature, based upon the owner's security policy. Authentication of the user is processed giving the user entitlement to applications and information granted by such hosted web services.

A variation on the public version of the persistent data control application may be implemented whereupon, once the persistent data control application is installed, that desktop/user may register with any other server using the public secured session to control secure data delivery or to secure a transaction over the Internet, such as ordering and paying for products. This feature of the persistent data control application includes a secured database on the remote apparatus, transparent to the user that retains information pertinent to all secure servers with which the desktop and the user have been registered and/or to which subscription has been granted for services employing a public secured session. In each secured session, the server's identity is unique, private, registered, and is secured on the user's desktop using a dynamic key that may be provided only by the primary secure server, thus providing for a unique secured session between the desktop and each server registered. The database is secured in such a fashion as to make it not transportable from the desktop on which it is installed. This database would contain all the unique information required to establish an immediate secured session between the host server and the known entity, such as the host's IP address and the desktop's registry information.

II. Secured Session Form 2: Private Internet Network

As an example, the second form of a secured session allows no public component to the Internet host site. Under such circumstances, because the persistent data control application is pre-registered with the server, the session can be instantiated immediately upon the secure exchange of the dynamic session key, or upon user authentication in the form required by the data owner's policy and the secure exchange of the dynamic session key. The first instance that the remote desktop connects with the host server, a secured session begins to be built. Only the unique identifier needs to be passed from the remote browser to the server to establish a secured session because the user is already a registered and known entity.

The persistent data control application can also serve additional functions on the desktop. For example, it might be embedded in or called by an application on the desktop and might be used as a network interface other than a browser to connect with the Internet and the secure server.

The secured session, in either its public or its private form, may be extended beyond the communications session between the server and the desktop. Data, applications, and resources on the desktop owned and/or controlled by the

server are or may be secured until the session has been established, at which time the unique key(s) required to gain access to, use, or render the data is passed to the desktop. Rules of accessing, using, and rendering the data are encoded into the data secured on the desktop and may only be overridden upon granting of permission by the server. A description of this feature is further detailed below under the heading "Remote Authorization for Rendering of Secured Data on a Remote Apparatus."

If the owner's policy allows, the persistent data control application installed on a desktop and licensed to a user may be ported to and installed on another desktop or apparatus. However, the following conditions will then apply: (1) none of the previously-secured data provided via a download and secured on the original desktop will be transportable to the new desktop; (2) all registrations and/or subscriptions with previous servers using a secured session must be renewed. Policy to deal with re-subscribing must be incorporated into each server's services, as defined by the owner's policy, so that at no time may there be a desktop license registered on two different desktops or to two different users. One of the advantages of the present invention is that it prevents fraudulent access to the data and protects the user in case there has been a theft of the system where the persistent data control application is installed.

III. Remote Authorization For Rendering Secured Data On A Remote Apparatus

A process wherein secured data can only be rendered, accessed, or used by a remote apparatus upon establishing a communication session with a control apparatus that provides information and/or the key(s) to unlock the secured data for rendering, accessing or using by the remote apparatus. The secured data may be of any type, comprising documents, control information, software programs, applications, images, video, music, and database information, etc. The process in its entirety, as described below, applies both to the control of secured data that is resident on or is downloaded via a communications medium to the user's remote apparatus, and to the control of data secured only on a distributed storage medium.

The process relies upon a secure communication methodology, e.g., a secured session, which may be standard or proprietary, between the control apparatus and the remote apparatus, such that the control apparatus grants the remote apparatus the rights to render, use, or access the secured data.

The process further incorporates a control apparatus that may be an administrative/authorization computer or similar apparatus that is not limited to any specific type or brand of computer or operating system and that has the functionality to perform all required tasks of the process comprising: (1) authorizing the remote

rendering, accessing, or using the secured data which may be resident on, downloaded to the remote apparatus, or stored on distributed media to be rendered on the remote apparatus; (2) interfacing with all required internal applications and databases necessary to provide such administrative components as data keys and such functionalities as subscriber validation and charges; (3) securing all communications with the remote user by the means specified and used by the control apparatus; and (4) communicating with the remote apparatus over a network, be it public, private, or proprietary in nature.

The administrative functions of the control apparatus further include the following: (1) tracking all secured data requested, distributed, authorized, rendered, used, and/or accessed by a remote apparatus; (2) consummating a transaction between a control apparatus and a remote apparatus; and (3) tracking all identifying data pertaining to a remote apparatus that is subscribed or known to the control apparatus and that has rights to the secured data. The process may also incorporate administrative functions that enable a remote apparatus to view, subscribe to and order the secured data, and whenever charges apply, to complete a secure financial transaction.

The process also comprises a remote apparatus, such as a computer or set-top box, that includes functions and capacities for: (1) accepting a distributed storage medium containing the secured data; (2) communicating securely with a control apparatus to which the remote apparatus has rights or subscribes, or with which it is authorized to communicate; (3) using the key(s) provided by a control apparatus to unlock the secured data for rendering, use, or access; (4) rendering, giving access to or using the secured data as prescribed by the control apparatus; and (5) interfacing with any and all input and output apparatus necessary for use or control.

The security for communication between the control apparatus and the remote apparatus may or may not be the same as the security used to secure the data on the storage medium to be rendered, accessed or used by the remote apparatus. The security of the process must include secure means of moving the key(s) to the remote apparatus to enable the rendering, accessing or using of the secured data by the remote apparatus and if required, a secure methodology for consummating any other form of transaction securely over a private or public communications medium, such as the Internet.

The security of the data persists, having used the persistent data control application throughout the process, except as the data is rendered, accessed, or used by the remote apparatus according to the policies and rules established by the owner of the data. The owner of the data dictates the rules and policy for

rendering, accessing, and using the secured data and, the remote apparatus has the means to enforce the rules at the time of rendering, accessing, and/or using the secured data. The rules and policy for rendering, accessing or using the secured data remain persistent as defined by the owner of the data regardless its control apparatus be at the secure server or at the remote apparatus. The rules and policy that may be dictated by the process required by the control apparatus, once the secured data has been rendered and accessed, comprise one or more functionalities, such as printing, copying, saving, or specifying an allotted time or a number of times that the secured data may be used or when the data may be rendered.

The process allows an open distribution of the secured data, such that, if a storage medium containing the secured data can be transported to another remote apparatus, that apparatus, being either a known subscriber or a new subscriber, may communicate with the control apparatus in order to be granted the rights to render, access, or use the secured data contained on the distributed storage medium. Thus, the secured data is apparatus- and subscriber- independent, but the control of the secured data remains with the control apparatus throughout the process described herein.

The following figures, Figs. 9-12, and their descriptions may incorporate or be preceded by standard methodologies to authenticate a user to the control apparatus or a remote apparatus to a control apparatus, and to the secure exchange required dynamic session keys.

Turning now to Figs. 9A-9B, one embodiment of a flow diagram for establishing a secured session with a registered user in accordance with the principles of the present invention. This method may be preceded by a standard authentication methodology for user or apparatus and transfer of a session dynamic key. In Fig. 9A, a remote apparatus 174 calls the system to request a secured session and transmittal of data in block 176. Then, a unique identifier is encoded with a level 1 encode key and sent to a server 182 in block 178. A level 1 encode may use a custom key of the Virtual Private Network (VPN) or a time stamp if a public secured session to encode is requested. The remote apparatus subsequently awaits return status from the server 182 in block 180.

The server 182 parses the data packet or HTML for an identifier on extension and decodes the identifier in block 184. The server calls the secure server and decodes the data in block 186. Then, a call to a registry component is made and the unique identifier is validated in block 188. If the user is not valid in block 190, i.e. the "no" path, a call to a security audit and a trace component is made in block 192 in order to trace and log an illegal remote session, and the session is terminated in block 194.

If a valid user is established in block 190, i.e. the "yes" path, the server looks up the encode keys for the remote user on the unique identifier in the registry in block 196. The keys are then passed to the secured session for all future session encoding in block 198. The server then initiates building of a secured session for the remote user in block 200. Next, a request for user authentication is generated in block 202, and a call to a secured session and encode is made in block 204. Subsequently, the server sends to the remote user an encoded request for user identification or password in block 206.

Next, the remote apparatus decodes the server request using level 2 user keys in block 208. In one embodiment, the level 2 encode uses three keys out of four for encoding purposes. The password or digital signature is then entered in block 210. Then, the remote apparatus determines whether the password or signature is valid in block 212. The remote apparatus then performs either a desktop validation check and terminates the session in block 214 or proceeds to encode a password or signature using level 3 encode in block 216. The level 3 encode uses the password/signature as a fourth key for the secured session component to complete the secured session on desktop encoding in block. The encoded password or signature is then sent for authentication to the server in block 218. The process continues on in Fig. 9B.

In Fig. 9B, after the remote apparatus sends the encoded password or signature for authentication to the server, the server parses the encoded password or signature and passes the received data to the secure server in block 220. The secure server 220 is called and decoded on Level 3 keys in block 222. A call to a user authentication component is then made in block 224, and the password or signature is validated in block 226. If the password or signature is not valid, i.e. the "no" path, a call is placed to a security audit and trace components to trace and log an illegal remote session in block 228. Then, the session is terminated in block 230.

If a valid password or signature is received from block 226, i.e. the "yes" path, the secure server is authorized in block 232, and in block 234, the final key is passed to the persistent data control application for all future secure server encoding. A complete generation of a secured session for a remote user on the server is then made in block 236. The status is generated and the server is ready for services requested from the remote user in block 238. A call to the secure server and encode is requested in block 240, and the encoded status is sent to the remote user in block 242.

Then, the remote apparatus 174 decodes the data packet or HTML using all remote user keys and authorization in block 244. Next in block 246, the status is validated, and the secure session is set as complete. A request message is

then generated in block 248, and encoded on all keys, in block 250, with a level 3 encode by using all keys known to the secure server component for the secure server encoding. The remote apparatus then sends the encoded request to the server for further processing in block 252.

Figs. 10A-10F are functional block diagrams of various embodiments illustrating a method of registering and establishing a secured session with a new registered user in accordance with the principles of the present invention. This method may be preceded by a standard authentication methodology for user or apparatus and transfer of a session dynamic key. In Fig. 10A, at a client secure application or browser 254 initiates a session and encodes a unique ID, which is sent to a secure server 256 through a communications network 258, e.g., the Internet. The secure server 256 decodes the unique ID and searches for the ID in a subscriber registry database 260 for a license key. The secure server 256 then initiates the generation of a user secured session on the secure server 256.

As shown in Fig. 10B, the secure server 256 encodes and sends a requests for a unique apparatus key to the client secure application where the client secure application or browser 254 is located through the communications network 258. The client secure application or browser 254 decodes, and the request for a unique apparatus key is then processed.

As shown in Fig. 10C, the client secure application or browser 254 encodes a unique apparatus key which is sent to the secure server 256 through the communications network 258. The secure server 256 then decodes and passes the unique apparatus ID to the registry database 260. The secure server 256 continues to build a user secured session on the secure server 256. Subsequently, the subscriber registry database 260 is searched for the ID and is updated with the unique apparatus key.

As shown in Fig. 10D, the secure server 256 encodes a session status and requests authorization and sends it to the client secure application through the communications network 258. The client secure application or browser 254 then decodes and processes the session status and requests authentication.

As shown in Fig. 10E, the user enters a password/authorization code which is then encoded at the client secure application 256 and is sent from the client secure application 256 through the communications network 258 to the secure server 256. At the secure server 256, a decode is performed, and the password or authorization is passed to an authorization server 262.

As shown in Fig. 10F, the authentication status is passed to the secure server 256 and is encoded. The session status is completed and sent through the

communications network 258 to the client secure application that then decodes. The process session status is then complete.

Figs. 11A-11B are functional block diagrams of various embodiments illustrating a method of requesting for specific content or data key and rendering in accordance with the principles of the present invention. This method may be preceded by a standard authentication methodology for user or apparatus and transfer of a session dynamic key. In Fig. 11A, the client secure application or browser 254 requests an authorization to encode for the unique data ID which is sent to the secure server 256 via the communications network 258. At the secure server 256, a decode is performed, the subscriber is verified from the subscriber registry database 260, and the data and subscriber ID are passed onto a data application server 264. The data application server 264 makes a query to a database 266 for verification of account with a subscriber usage database 268, and for information such as applicable charges. The data application server 264 also obtains authorization keys for the data from the database 266 if the account is verified.

In Fig. 11B, the data application server 264 sends the authorization keys to the secure server 256. The secure server 256 encodes data and the data keys or just the data keys and sends the data and/or the data keys to the client secure application or browser 254 for rendering.

Figs. 12A-12D are functional block diagrams of various embodiments illustrating a method of establishing a secured session with a registered user in accordance with the principles of the present invention. This method may be preceded by a standard authentication methodology for user or apparatus and transfer of a session dynamic key. In Fig. 12A, a secured session is initiated on a client secure application or browser 270. A unique ID is encoded and sent to a secure server 272 through a communications network 274. The secure server 272 decodes the unique ID, searches on the ID in a subscriber registry database 276 and initiates the generation of a user secured session on the secure server 272.

As shown in Fig. 12B, the secure server 272 encodes the session status on and sends a request authentication through the communications network 274 to the client secure application where the client secure application or browser 270 is located. The secure application or browser then decodes, processes the session status, and requests authorization.

As shown in Fig. 12C, a user password authorization code is entered and encoded at the client secure application. The encoded password/authorization code is then sent through the communications network 274 to the secure server 272.

At the secure server 272, the encoded password/authorization code is decoded, and the password or authorization code is then passed to an authorization server 276.

As shown in Fig. 12D, the authentication status is passed to the secure server 272, is encoded session status complete and sent through the communication network 274 to the client secure application or browser 270. At the client secure application or browser 270, the authentication status is decoded, and the session status complete is then processed.

The following are examples of the implementation of a persistent data control system. It is appreciated that other implementations can be used without departing from the present invention.

One example of the implementation is that a persistent data control system in accordance with the present invention is a component of a hosted web service and a client browser. The hosted web site having a secure server has access to all subscriber authentication, profile, access rights and usage databases. Accordingly, the data is encoded according to the use and access rights of a particular subscriber using encoding keys for a specified user to build a secure session and for a particular data type as necessary. User and apparatus authentication to the server may occur in any standard or customized manner deemed necessary by the installation hosting the web content or services. The URL markup language and other referenced content of a web page requested by the subscriber may be encoded as a single file or individually as per implementation of the persistent data control system on a hosted server that forms a secure server in accordance with this invention. The persistent data control system embedded in an end user's browser, i.e. the secure client browser, decodes the encoded access and/or use rules embedded within the encoded file and/or data type. The data is then rendered according to the use rules embedded within the encoded file and/or data type. As an example, the rules may specify that at no time will any of the web page or its referenced content be stored on the device. The browser will therefore be disabled from allowing the user to print, copy or store such content in the Internet Temporary Folder, as is customary or in any other folder as may be desirable by the user. Furthermore, the web page may be transactional in nature and require a simple response, change, or entry of one or more data fields where, upon a response from the user, the secure client browser encodes such data according to the rules embedded within the encoded web page using the appropriate session and data keys and return such encoded data to the server. The server then decodes such information using the appropriate keys and process the decoded data in accordance with its application.

In the foregoing example, one of the encoded data types of the encoded file may be a database having a specific structure and an image or multiple images that are part or referenced in the database. An example of such encoded database and referenced images is a patient DICOM medical record. Being desirable to use the Internet for transmitting a patient DICOM medical record and the browser to render the encoded data types, the persistent data control system in accordance with the present invention can control the access to the data elements within the patient DICOM medical record and the images and their use according to the access and use rights encoded with the file or each data type independently. Thus, a patient DICOM medical record requiring that at no time any part of it be separated from the whole, the complete record may be sent to various users whereupon each user of a different user group may only gain access and use the data according to its user group access and use rights mapped into the encoded file or data type. Such access and use rights are enforced by the secure client browser upon rendering the data by the browser. The above examples may be implemented by a client application having the access by programmatically calling the persistent data control system, or may be implemented by having the persistent data control system be embedded within the client application itself.

Yet another exemplary implementation of the present invention is for the purpose of securing e–mail and its content according to the rules of the owner of the data whether the data originates from a server or a user. The persistent data control system may be embedded in a user's e–mail application. The originator of an e–mail may specify the access and use rights for the body of the e–mail as well as any attachments of the e–mail. The e–mail may be sent to another user having the persistent data control system incorporated into their e–mail service and render the body and attachments of the e–mail in accordance with the rules defined by the originator of the e–mail. Such rules persist for the life of the e–mail and in the manner defined. This example may be implemented in a variety of ways. One such manner of implementation utilizes a secure server to arbitrate e–mail movement wherein the body of an encoded e–mail may be decoded by a secure server using session keys of an originator. The same body of the e–mail is encoded with recipient's session keys. The attachments may be left encoded because the keys to them are embedded within the encoded control information of the encoded file, or because the originator may require the recipient to request the keys from the originator or from the secure server at the time the e–mail and its attachments are to be rendered. Other implementations may be utilized without departing from the spirit of this invention.

IV. SECURITY SOFTWARE APPLICATION

The security software application in accordance with the present invention provides a method of encoding and decoding digital data. The security software application provides an encoding mechanism via a random number generator for all possible character sets and a program or logic means for scrambling the information such that no character are represented by itself or reside in its original position. The security software application employs one or more random number generator keys in a manner that prevents the data from being decoded on any apparatus other than the one targeted. The security software application may be incorporated and/or embedded into other applications or systems.

The owner of the data can extend and enforce the policies and rules that govern and control the data and its life cycle to a targeted recipient of the data. Specific data controls that may be granted and enforced include the ability to read, write, copy and/or print, the term the data are retained, and whether the data are retained on the apparatus in a secure or open form. The security software application secures the data to any apparatus having a unique identifier and allows access to the unique identifier. The application may also lock the data to an individual, wherein means for authenticating the individual's identity is imposed or required. Program or logic means of authentication include passwords, biometrics, certificate authority, and/or digital signatures, etc.

The mechanism for encoding and decoding to and from a buffer or file facilitates control over the rendered data and the applications in a given operating environment. The applications that utilize data streaming, such as music or movies, over a network may utilize a buffer-to-buffer or file-to-buffer feature to ensure the security of the data over the network as well as to prevent its capture and copying.

The security software application relates generally to a method of securing digital data, wherein any file type or data stream type can be secured with or without the use of a standard encryption algorithm. The file type or data type may comprise documents, control information, software programs, applications, images, video, music, database information, and any other digitized analog information of any length. The method in accordance with the present invention does not increase the size of the original information substantially. For example, the method according to the present invention merely adds an encrypted header to the original file. The encrypted header does not increase the original file size by a significant amount, usually no more than 1500 bytes. Furthermore, the method may be applied to digital streams comprising video or voice streaming. Moreover, the

encoding process can be re-applied to the same data multiple times to increase the degree of security.

In another embodiment of the present invention, the data is encrypted and formatted in a database file format. The file includes a header that has a plurality of fields. At least one of the fields defines the data's persistent control policy that controls use rights of a recipient. The data persistent control policy is granted by the owner of the data.

In accordance with the principles of the present invention, the method may generate a single file type that is verifiable to prevent attacks and spoofing of the encoded data. For example, the single encoded file type may be checked at a firewall or a proxy to validate the data before allowing them to enter into the system and decoded to prevent unauthorized access or attacks on the system.

The method according to the security software application also allows the owner of the data to define the policies or rules for rendering, accessing, and using the encoded data. Such policies or rules are a part of the encoding scheme and data and, are enforced when the recipient receives and decodes the data. The method in accordance with the invention further provides multiple key schemes, a method to define and control the use of the keys, and the encoding and decoding logic. The method in accordance with the invention also can prevent decoding of the data except on a specific apparatus and by a specific person and software installation.

A process by which digital data secured/encoded and unsecured/decoded by incorporating: (1) means for organizing the digital information; (2) logic or program means for inputting the data to the encoding process from a buffer, a stream, or a file, and logic or program means for outputting the data to a buffer as a stream or a file; (3) the logic or interfaces to incorporate any standard or customized encryption and decryption logic; (4) a key template for encoding the data types or the composite file; (5) logic or program means of building a master seed from unique sub-keys used by the random number generator; (6) logic or program means for encoding the control information for the data types or composite file; (7) logic or program means for encoding the rules of rendering, accessing, and using the data; (8) logic or program means for encoding and decoding the data types; (9) logic or program means for enforcing the rendering rules; (10) logic or program means for establishing the use and source of keys for encoding and decoding and the establishing the process flow of the encoding and decoding process; (11) logic or program means for decoding that is in effect the reverse process of encoding; (12) means for determining and rendering the data useless if the secured information has been altered in any way; (13) means for allowing a definition of how and from where the data is input and output from the encoding and

decoding process; and (14) logic or program means for encoding multiple files and their encoded headers and concatenating multiple files into a composite searchable encoded file.

The encode and decode process preferably comprises the following components: headers, file encode, file decode, buffer encode, buffer decode, encode/decode templates, key, rendering rules, rendering, process logic and level flow. Each of these components is described below in detail.

A. Headers

As described above, the data control elements can be included in a header or other parts of the composite file. For simplicity and illustration, a header is used herewith.

A composite file header is generated for the complete encoded file and for individual data types and is comprised of the pointers to each individual encoded data type and information that allows the process to control its logic and encode level. The logic and level flow information defines what key or key set are used to decode and how the decoding process occurs. The process flow is set by the process itself in a networked environment or programmatically to enable the implementation of one set of source code in potentially many different environments.

Each encoded data type may have its own encoded header of a variable length and comprises process control information. The encoded header includes the length of the encoded digital information, the length of the original file, the original file name, and the type extension. Other information within the header comprises a dynamic key and its change status, a set of rendering rules, a date of creation. This header also contains a set of rendering rules to include but not limited to, an expiration date for the rendering of the data and a counter and decrementor for controlling the number of times the data can be rendered, accessed or otherwise used.

B. Data Encode

Data are read and encoded in accordance with the standard or customized encryption algorithm being utilized. The secure application implementing the encryption algorithm incorporates a mechanism enabling and for identifying whether the source of the information to be encoded from a buffer or a file is provided. A bit is set in the encrypted header for identifying the data source and defines how the input of the information is to be handled during the file encode process, and the lengths are set in the encrypted header. This is due to the fact that

data from a buffer may be streamed and of an indeterminable length until the last byte is read. Alternatively, a file has a fixed length.

C. Data Decode

Data from the encoded header is decrypted and used to initialize the key and the dictionary and other related processing. Segments are read from the encoded data segment in the same manner as described in File Encode. The mechanism for identifying whether the output of the information to be decoded to a buffer or a file is provided. A bit is set in the encrypted header at the time of the file encode process to define to the file decode process how the output of the information is to be handled. Data may be sent to a buffer or a file and then stored or rendered by an application or viewer. Output to a buffer prevents any intermediary or permanent file from being created and provides greater control of the decoded data. The output of the decode process is written to a file of the same length as the original input data file along with its original file extension.

D. Keys & Seed

One or more keys may be used to create the encode/decode key(s). This process may incorporate multiple keys that can be used singularly, and in various combinations, dependent upon the logic and encode level flow, and keys may further be encoded into the header. Any combination of the keys and in various combinations, dependent upon the logic and encode level flow fixed or dynamic keys may be used for the encode and decode of all headers and data.

Generally, the term "key" and the term "seed" are interchangeably used. The term "compound key" and the term "key set" are also interchangeably used.

The following describes an exemplary source and use of the keys but is not limited to such. The length of the keys may vary from 4 to 32 characters/bytes in accordance with the encryption method. The values of the keys are acted upon to create a single value from which no less than 32 bits are extracted from some random portion that is defined by the program and is used as the key for the encryption process.

The first key is global and dynamic and is stored in the encoded header of the file. This key dynamically generated by the secure server application for a user session and is always transmitted in a secure fashion using any standard or custom methodology to insure its security. Furthermore the dynamic key is stored only in random memory and never stored on a permanent storage medium.

The second key is the unique license number of the secure software installed on the apparatus. In a network environment, this key is registered on the client apparatus at the time the client secure application software is installed and on the server subscriber database. This key is accessible to the server for encode and decode by requesting and receiving the client ID that is associated with the licensed software distributed to the client. The client ID is used to index the database for the license key.

The third key is a unique number of the apparatus upon which the client secure application is installed. This key is retrieved from the apparatus each time it is to encode or decode data. In a network environment where two or more apparatuses exchange encoded information, the key are stored in the server subscriber database and accessible by using the client ID in the same format as the license key. This key is passed once, upon the initial registration of the user or client, using the dynamic and the license keys to secure the keys so as to protect the keys in transmission, and is then placed into the server subscriber database for later retrieval and use.

The fourth key is a dynamic key that can be used for a password, digital signature, some other user identifying or authenticating mechanism, or any other required system or user definable key.

The keys are processed together to generate a compound seed, e.g. a master key, which is fed to the encryption algorithm for purpose of encoding and decoding of all header and data in accordance with the encode logic and flow.

E. Rules Of Rendering

The means for the owner to establish the rules or policies for rendering, accessing, and/or using the encoded information and the means for these rules to be encoded into the file header. These rules incorporate: (a) control of how the data are saved on the decoding apparatus; (b) if the data is to be retained as an encoded file and whether the data may be printed, displayed or saved as an open file; (c) the number of times the encoded data may be viewed before the rendering process erases the encoded file; (d) a length of time or days the encoded file is retained for viewing before the rendering process erases or destroys the information; and/or (e) how the data is to be viewed or rendered.

F. Rendering

A programmatic means to pass the rules to and control a rendering component by which they are enforced. The secure component of the present invention incorporates a default rendering engine that monitors, updates, enforces

the rules for use of the data such as text, image, audio, image, and video data when control of an external rendering application is not available to enforce the rules. Furthermore, the rendering component also provides the interfaces necessary to be implemented within an application and enforce the rendering rules. The encoded data is decoded to a memory buffer from which it is rendered to the printer, display device, or any other output device where the controls are available to prevent the ability of the recipient to save the data to any other file format outside the secured format. Once the rendering component has determined the rules governing the number of uses or expiration date have expired, and if and how the secured data may be saved on the rendering apparatus, the secured file is erased from the apparatus or the storage medium upon which the secured file resides, or allowed to be stored in an open decrypted format or in it encrypted format in accordance with the rules and policies incorporated in the encrypted header or the encrypted data.

The interfaces and the adaptability exist such that upon knowing the interfaces that control a rendering application, the required control to enforce the rules can be applied in any application. This is extremely applicable to players for video or music currently being moved over the Internet.

G. Logic And Encode Level Flow

The encode and decode process is made up of components that can be controlled programmatically or be set based upon how it is to be used by a system or the application that may call it or in which it may be embedded. The following encode level setting determines the flow through the process, use of the keys, and conditions for encode and decode.

Level 1 uses the dynamic key for encode and decode.

Level 2 uses the dynamic, license, and apparatus keys.

Level 3 incorporates and provides the interface for the dynamic key to be input and used.

Level 4 uses the dynamic key and is connected to a server that provides a unique dynamic key for encode. At the time of decode, the recipient is connected to the server, which provides the dynamic key to enable decode.

It is appreciated that additional levels may be used and can be reserved for expansion and customization as necessary.

Having described the present invention in the exemplary embodiments, modifications and equivalents may occur to one skilled in the art. It is intended that such modifications and equivalents shall be included within the scope of the claims which are appended hereto.

CLAIMS

What is claimed is:

1. A method of securely distributing data on a network, comprising the steps of:

a) providing an encoded file of a single file type having a plurality of file control fields, the file having at least one data type; and

b) incorporating at least one encoded use right into one of the control fields of the at least one data type.

2. The method of claim 1, wherein the step a) is performed at an application level.

3. The method of claim 1, wherein the method is capable of being embedded in an application which originates the at least one data type.

4. The method of claim 1, wherein the method is called by an application.

5. The method of claim 1, further comprising the step of:

c) incorporating multiple encoded use rights into the control fields of the at least one data type.

6. The method of claim 1, further comprising the step of:

c) incorporating at least one encoded access right into one of the control fields of the at least one data type.

7. The method of claim 5, further comprising the step of:

d) incorporating at least one encoded access right into one of the control fields of the at least one data type.

8. The method of claim 1, wherein the method is performed in a distributed network environment.

9. The method of claim 1, wherein the method is performed in the Internet environment.

10. The method of claim 1, wherein the method is performed in an Intranet environment.

11. The method of claim 1, wherein the encoded use right is encoded with the at least one data type.

12. The method of claim 1, wherein the encoded use right is encoded independently from the at least one data type.

13. The method of claim 1, further comprising the step of:

c) decoding the plurality of file control fields including a file control field for the at least one encoded use right.

14. The method of claim 13, further comprising the step of:

d) decoding the at least one data type.

15. The method of claim 14, further comprising the step of:

e) rendering the decoded data type in accordance with the decoded use right.

16. The method of claim 6, further comprising the step of:

d) decoding the plurality of file control fields including a file control field for the at least one encoded use right.

17. The method of claim 16, further comprising the step of:

e) decoding the plurality of the file control fields including a file control field for the at least one encoded access right.

18. The method of claim 17, further comprising the step of:

f) decoding the at least one data type in accordance with the decoded access right.

19. The method of claim 18, further comprising the step of:

g) rendering the decoded data type in accordance with the decoded use right.

20. A system for securely distributing data on a network, comprising:

an encoded file of a single file type having a plurality of file control fields, the file having at least one data type; and

means for incorporating at least one encoded use right into one of the control fields of the at least one data type.

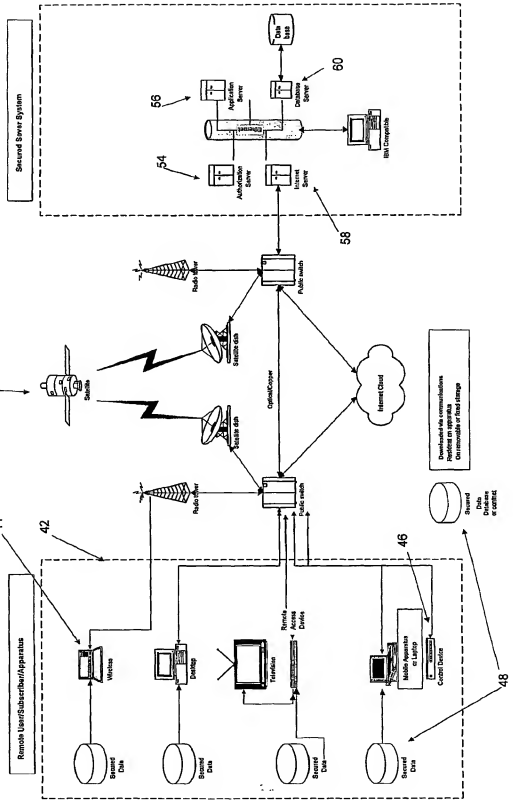


FIG. 1

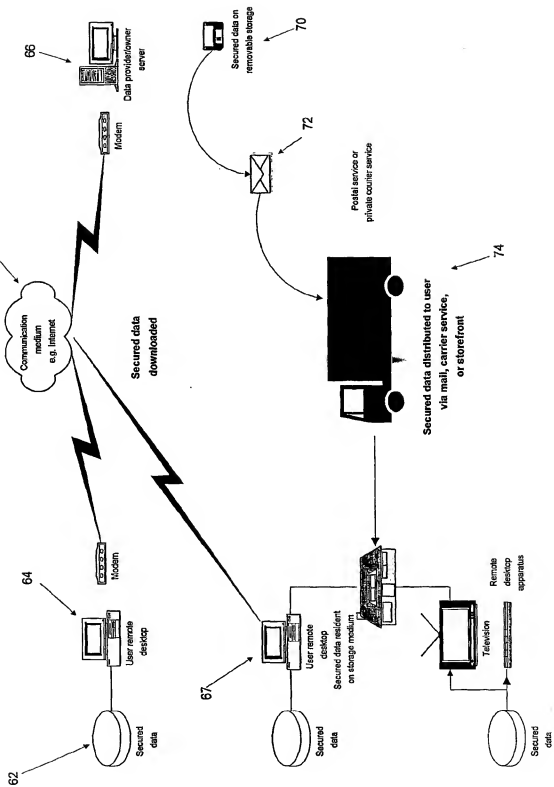

66
Data provider/owner server
Modem
68
Communication medium e.g. Internet
Secured data downloaded
Modem
64
User remote desktop
Secured data
62
67
User remote desktop
Secured data resident on storage medium
Secured data
Secured data on removable storage
70
72
Postal service or private courier service
74
Secured data distributed to user via mail, carrier service, or storefront
Remote desktop apparatus
Television
Secured data

FIG. 2

FIG. 3

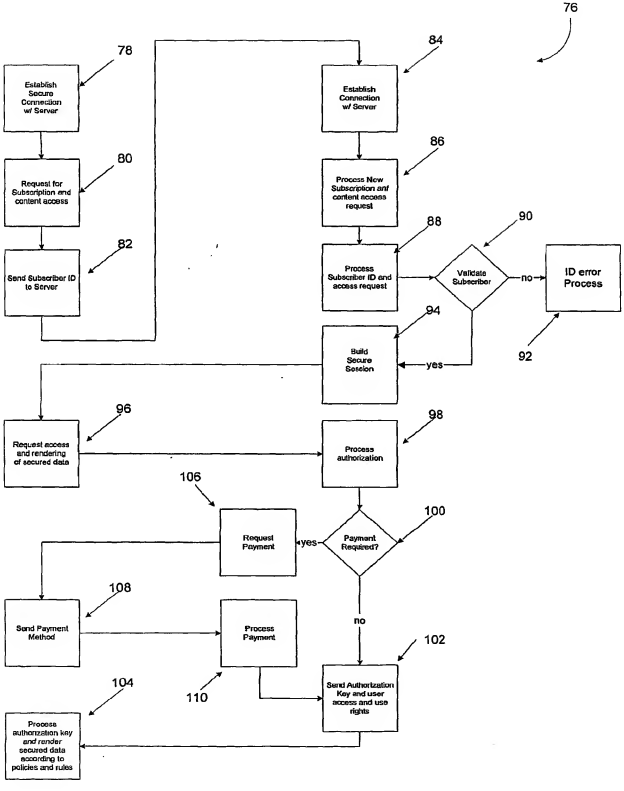


FIG. 4

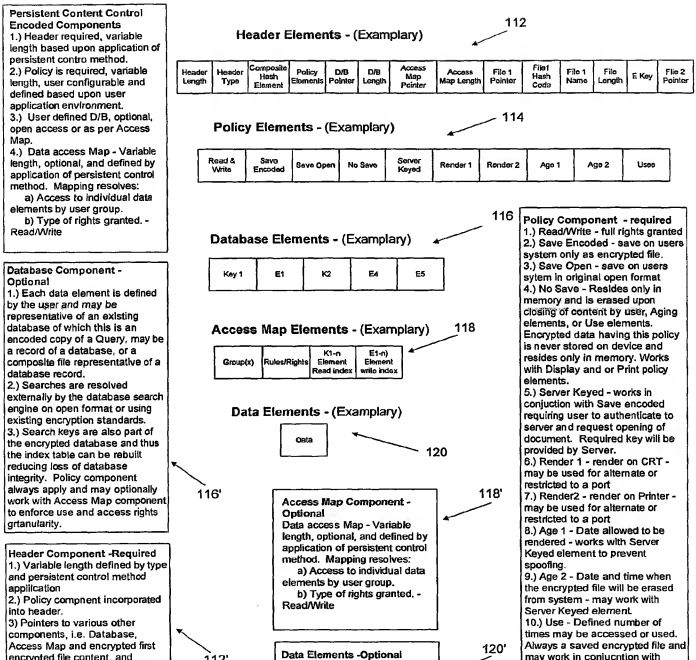

Persistent Content Control Encoded Components
1.) Header required, variable length based upon application of persistent contro method.
2.) Policy is required, variable length, user configurable and defined based upon user application environment.
3.) User defined D/B, optional, open access or as per Access Map.
4.) Data access Map - Variable length, optional, and defined by application of persistent control method. Mapping resolves:
a) Access to individual data elements by user group.
b) Type of rights granted. - Read/Write
Header Elements - (Examplary)
112
Header Length
Header Type
Composite Hash Element
Policy Elements
D/B Pointer
D/B Length
Access Map Pointer
Access Map Length
File 1 Pointer
First Hash Code
File 1 Name
File Length
E Key
File 2 Pointer
Policy Elements - (Examplary)
114
Read & Write
Save Encoded
Save Open
No Save
Server Keyed
Render 1
Render 2
Age 1
Age 2
Uses
116
Database Elements - (Examplary)
Key 1
E1
K2
E4
E5
Policy Component - required
1.) Read/Write - full rights granted
2.) Save Encoded - save on users system only as encrypted file.
3.) Save Open - save on users system in original open format
4.) No Save - Resides only in memory and is erased upon closing of content by usar, Aging elements, or Use elements. Encrypted data having this policy is never stored on device and resides only in memory. Works with Display and or Print policy elements.
5.) Server Keyed - works in conjuction with Save encoded requiring user to authenticate to server and request opening of document. Required key will be provided by Server.
6.) Render 1 - render on CRT - may be used for alternate or restricted to a port
7.) Render2 - render on Printer - may be used for alternate or restricted to a port
8.) Age 1 - Date allowed to be rendered - works with Server Keyed element to prevent spoofing.
9.) Age 2 - Date and time when the encrypted file will be erased from system - may work with Server Keyed element.
10.) Use - Defined number of times may be accessed or used. Always a saved encrypted file and may work in conjucntion with other policy elements.
Access Map Elements - (Examplary)
118
Group(x)
Rules/Rights
K1-n Element Read index
E1-n) Element write indexx
Data Elements - (Examplary)
Data
120
Database Component - Optional
1.) Each data element is defined by the user and may be representative of an existing database of which this is an encoded copy of a Query, may be a record of a database, or a composite file representative of a database record.
2.) Searches are resolved externally by the database search engine on open format or using existing encryption standards.
3.) Search keys are also part of the encrypted database and thus the index table can be rebuilt reducing loss of database integrity. Policy component always apply and may optionally work with Access Map component to enforce use and access rights grtanularity.
116'
Access Map Component - Optional
Data access Map - Variable length, optional, and defined by application of persistent control method. Mapping resolves:
a) Access to individual data elements by user group.
b) Type of rights granted. - Read/Write
118'
Header Component -Required
1.) Variable length defined by type and persistent control method applilcation
2.) Policy compnent incorporated into header.
3) Pointers to various other components, i.e. Database, Access Map and encrypted first encrypted file content, and possibly next encrypted file content which will be componsed of a header and other compnents as required to include another database and Access Map.
4.) A special key element may be encoded for accessing database and other encrypted file content.
112'
Data Elements -Optional
One or more Data elements may exist depending on header type. Digital data may be of any type and length. Data may also be streamed from one source to another encrypted from file to buffer or buffer to buffer methodology.
120'
114'

FIG. 5

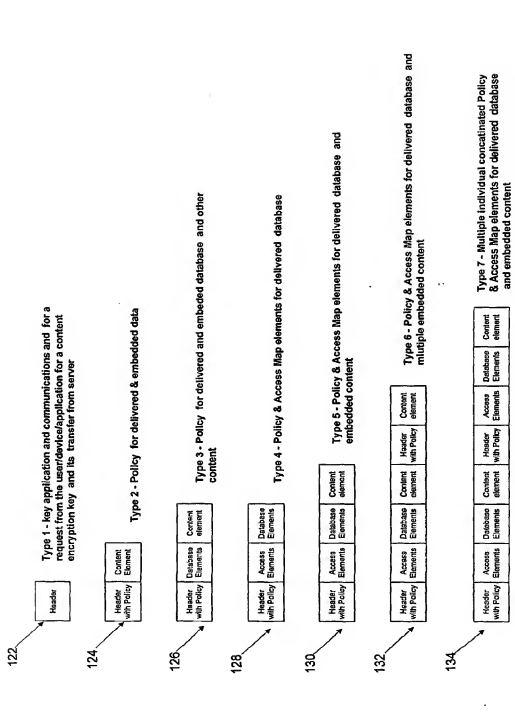


FIG. 6

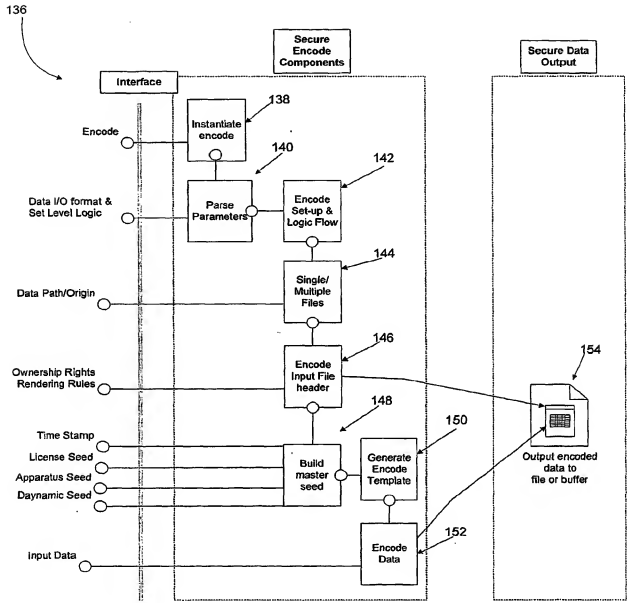
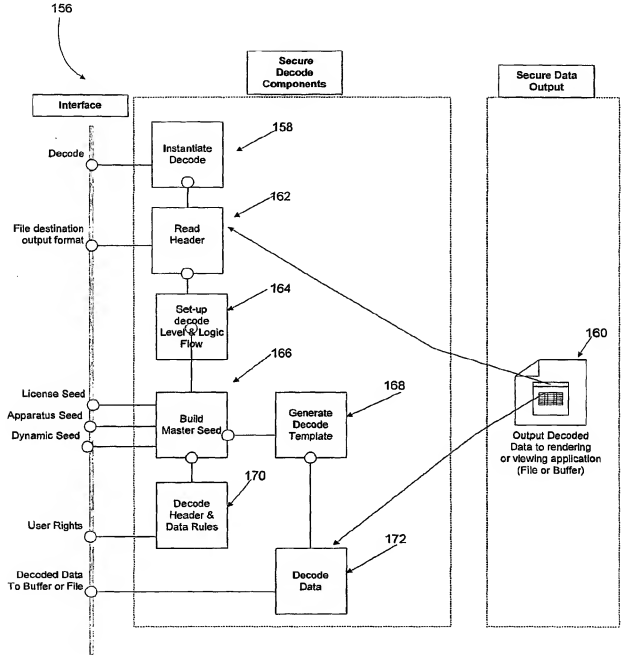


FIG. 7



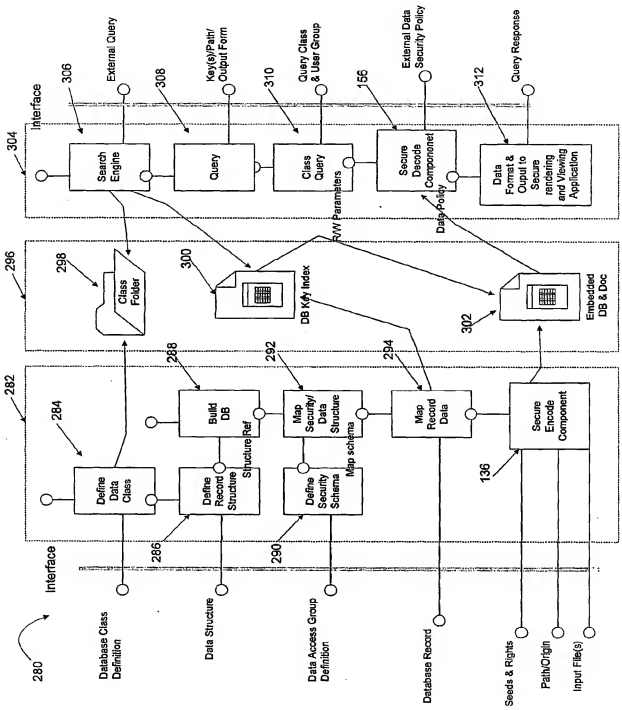


FIG. 8

FIG. 9A

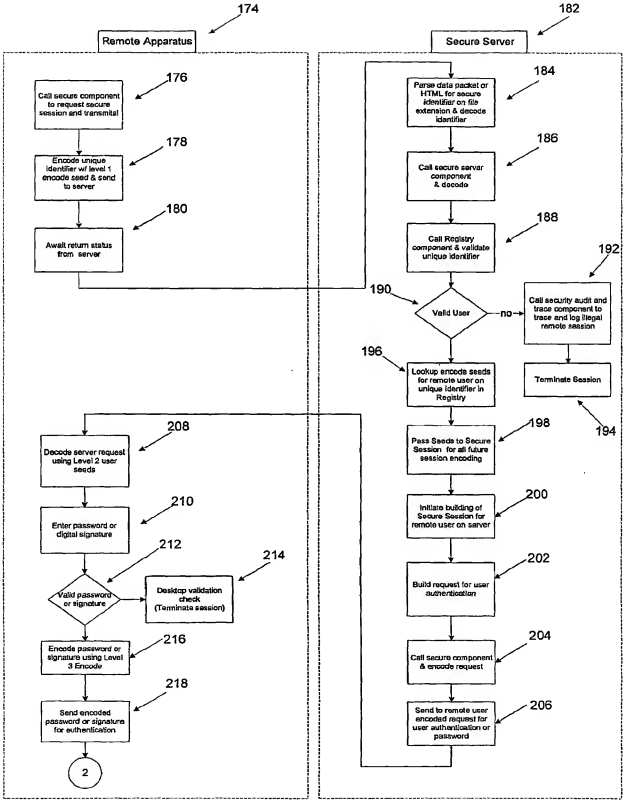


FIG. 9B

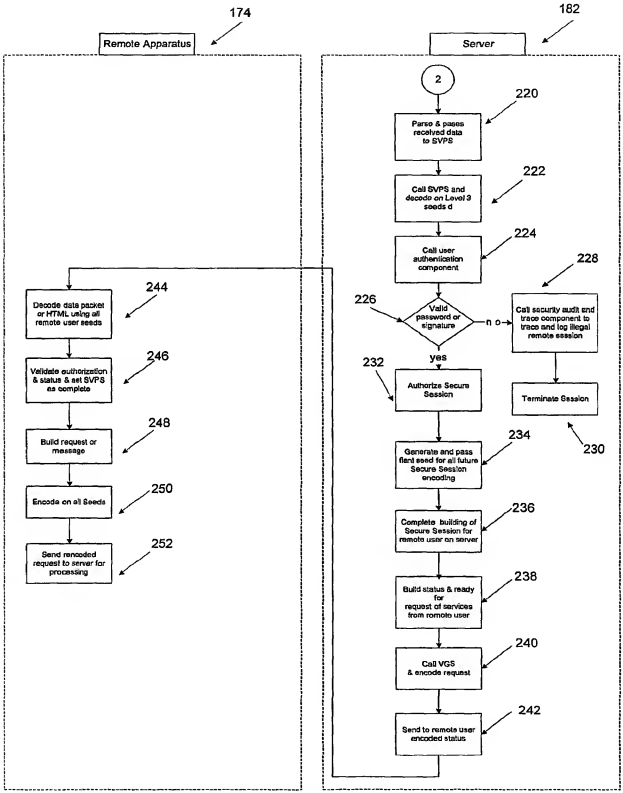


FIG. 10A

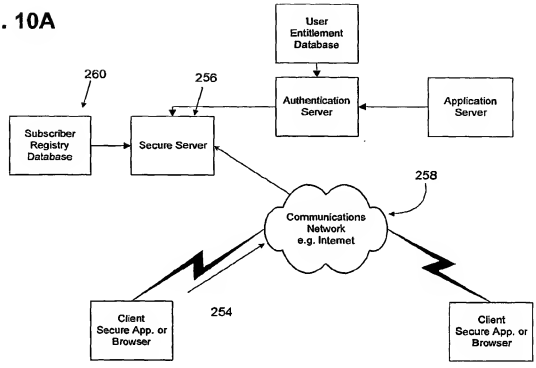


FIG. 10B

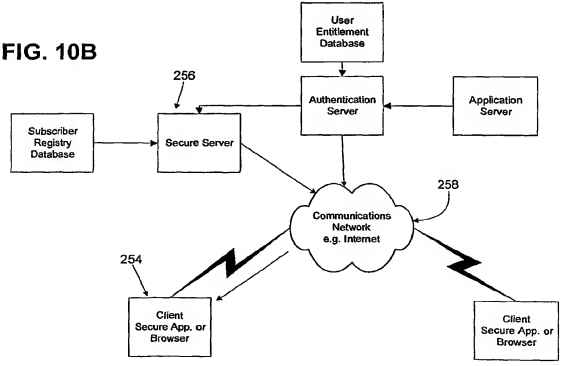


FIG. 10C

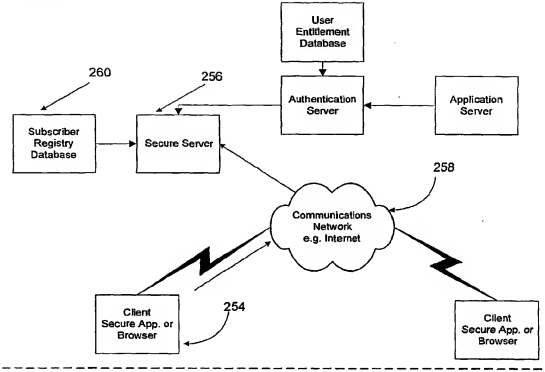


FIG. 10D

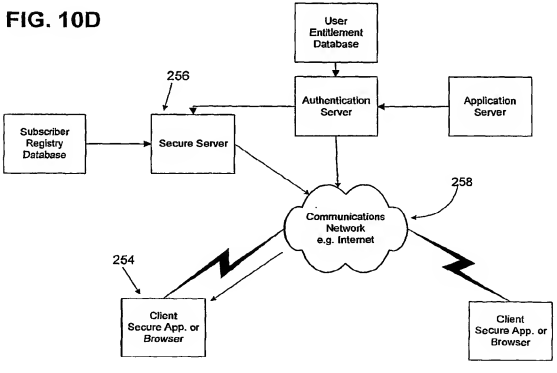


FIG. 10E

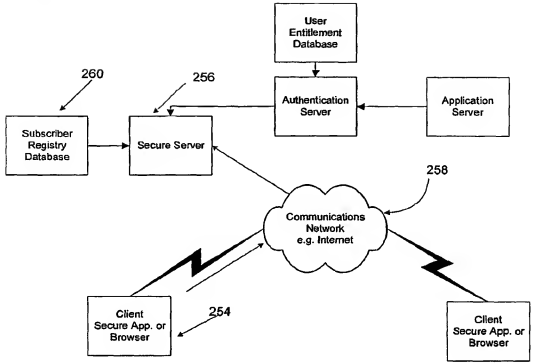


FIG. 10F

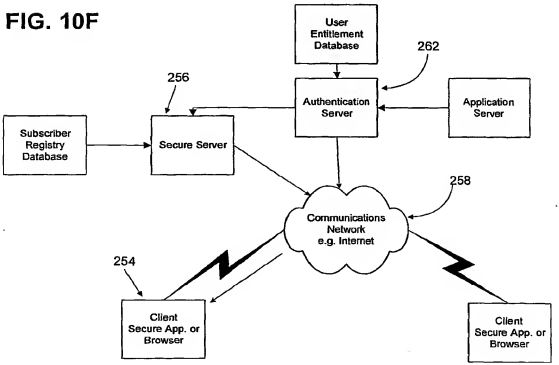


FIG. 11A

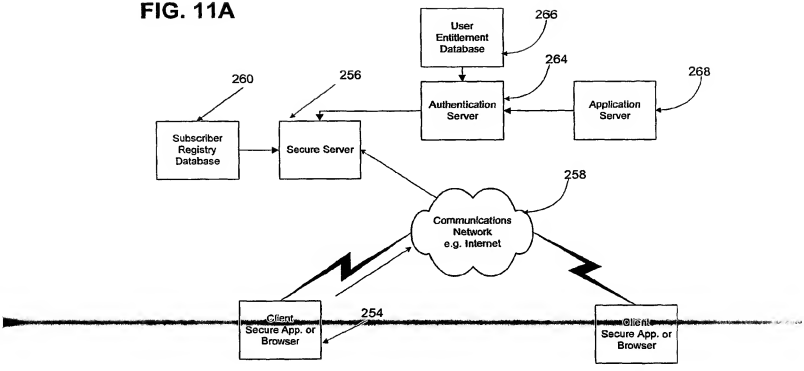


FIG. 11B

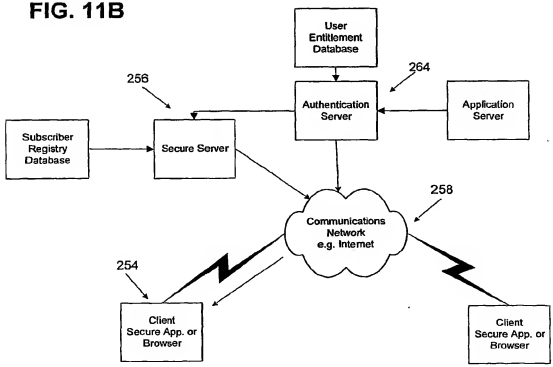


FIG. 12A

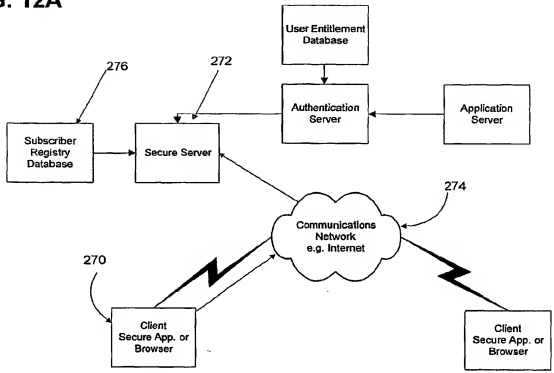


FIG. 12B

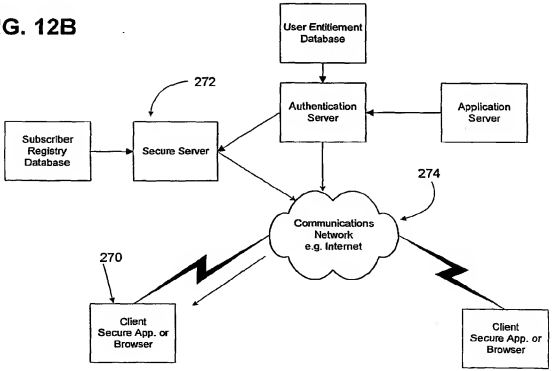


FIG. 12C

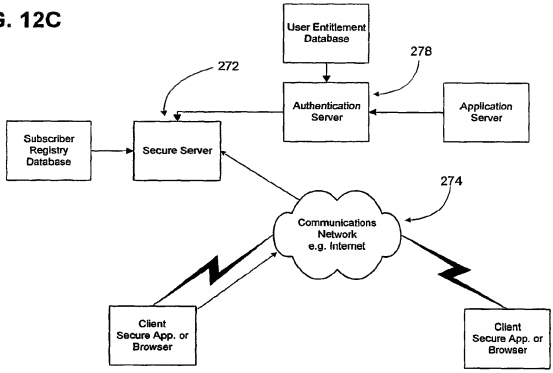
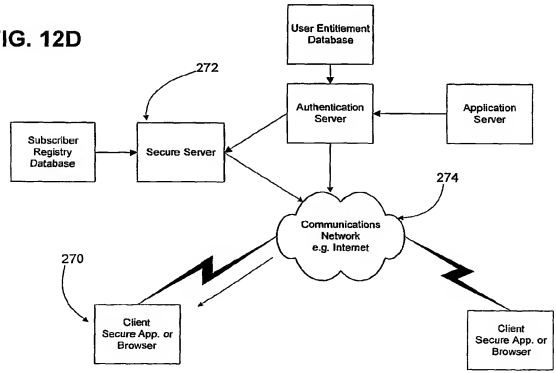


FIG. 12D



(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 August 2001 (30.08.2001)

PCT

(10) International Publication Number
WO 01/63387 A3

(51) International Patent Classification[7]: G06F 1/00

(21) International Application Number: PCT/US01/05505

(22) International Filing Date: 22 February 2001 (22.02.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/184,074	22 February 2000 (22.02.2000)	US
60/184,075	22 February 2000 (22.02.2000)	US
60/184,079	22 February 2000 (22.02.2000)	US

(71) Applicant: VISUALGOLD.COM, INC. [US/US]; Suite 203, 1011 First Street South, Hopkins, MN 55343 (US).

(72) Inventors: RICHARDS, Kenneth, W.; 7311 - 15th Avenue South, Richfield, MN 55423 (US). MURRAY, Arnold, E.; 10975 Stacy Trail, Chisago City, MN 55013 (US).

(74) Agent: BRUESS, Steven, C.; Merchant & Gould P.C., P.O. Box 2903, Minneapolis, MN 55402-0903 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AT (utility model), AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, CZ (utility model), DE, DE (utility model), DK, DK (utility model), DM, DZ, EE, EE (utility model), ES, FI, FI (utility model), GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SK (utility model), SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

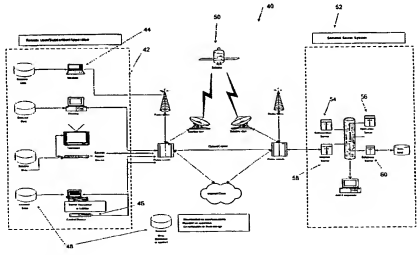
(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

(88) Date of publication of the international search report:
28 February 2002

[Continued on next page]

(54) Title: SECURE DISTRIBUTING SERVICES NETWORK SYSTEM AND METHOD THEREOF



(57) Abstract: A persistent data control system and method of securely distributing data on a network includes the steps of providing an encoded file of a single file type having a plurality of file control fields, the file having at least one data type, and incorporating at least one encoded use right and/or access right into one of the control fields of the at least one data type. The persistent data control method is performed at an application level, and is capable of being embedded in an application which originates the at least one data type or called by an application. The persistent data control method further comprises the steps of decoding the plurality of file control fields including the file control fields for the encoded use right and/or access right, decoding the at least one data type in accordance with the access right, and rendering the decoded data type in accordance with the decoded use and access right.

WO 01/63387 A3

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

INTERNATIONAL SEARCH REPORT

Intr tional Application No
PCI/US 01/05505

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F1/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data, PAJ

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	EP 0 778 512 A (SUN MICROSYSTEMS INC) 11 June 1997 (1997-06-11) abstract column 6, line 27 -column 13, line 57 figures 1-6	1-20
Y	EP 0 715 247 A (XEROX CORP) 5 June 1996 (1996-06-05) abstract page 3, line 44 -page 7, line 22 figures 5-10	1-20
A	US 5 765 152 A (ERICKSON JOHN S) 9 June 1998 (1998-06-09) abstract column 10, line 40 -column 27, line 24 figures 1,1A	1-20

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

A document defining the general state of the art which is not considered to be of particular relevance

E earlier document but published on or after the international filing date

L document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

O document referring to an oral disclosure, use, exhibition or other means

P document published prior to the international filing date but later than the priority date claimed

T later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

X document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

Y document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

& document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
21 November 2001	04/12/2001

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax. (+31-70) 340-3016	Jacobs, P

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Intr tional Application No
PCI/US 01/05505

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category ~	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 883 955 A (RONNING JOEL A) 16 March 1999 (1999-03-16) abstract column 3, line 22 -column 6, line 11 figure 4A	1-20

1

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inte tional Application No
PCI/US 01/05505

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
EP 0778512	A	11-06-1997	US	5708709 A	13-01-1998
			EP	0778512 A2	11-06-1997
			JP	9288575 A	04-11-1997
EP 0715247	A	05-06-1996	EP	0715247 A1	05-06-1996
			JP	8263438 A	11-10-1996
			US	2001023417 A1	20-09-2001
			US	2001010045 A1	26-07-2001
			US	2001014882 A1	16-08-2001
			US	6236971 B1	22-05-2001
US 5765152	A	09-06-1998	AU	7662496 A	30-04-1997
			WO	9714087 A1	17-04-1997
US 5883955	A	16-03-1999	NONE		

This Page Blank (uspto)